Python Stack Trace Interpretation

Paweł Rubin

September 2020

Introduction

The appended text file contains nine Python stack traces. The task is to examine the stack traces and provide a brief response for each one that summarizes what the problem or likely problem is, and the first line of code you should jump to in your code editor given the trace.

Overall analysis

It is easy to see that each presented problem has similar structure:

```
run_trace(i, lambda: f(op, arg1, arg2))
```

where `i` stands for the number of the task.

Then, it should be noted that `f` function in `lambda` stands for either of these three: `perform_calculation()`, `comp_calc()`, `calc_dict()`. Each of them performs an operation `calc()` on provided arguments.

With this in mind, let's dig into the first problem.

Traceback Problem 1

```
Traceback (most recent call last):
  File "stack_traces.py", line 36, in run_trace
    f()
  File "stack_traces.py", line 45, in <lambda>
    run_trace(1, lambda: perform_calculation(add, '1', 3))
  File "stack_traces.py", line 8, in perform_calculation
    calc(x, y)
  File "stack_traces.py", line 12, in add
    return x + y
TypeError: can only concatenate str (not "int") to str
```

As the `TypeError` message says: `int` cannot be concatenated to `str`. We can see that this operation was perfomed on the line 12 in the `x + y` expression in the `add()` function, which was called via `calc()` in `perform_calculation()`. The latter has been executed with `'1'` - a string, and `3` - an integer.

The problem here is the incompatibility of arguments. Depending on the desired result - we should change one of them accordingly. So the first line of code we should jumo to in owr editor is the line 45.

Traceback Problem 2

```
Traceback (most recent call last):
  File "stack_traces.py", line 36, in run_trace
    f()
  File "stack_traces.py", line 46, in <lambda>
    run_trace(2, lambda: perform_calculation(add, 7, '3'))
  File "stack_traces.py", line 8, in perform_calculation
    calc(x, y)
  File "stack_traces.py", line 12, in add
    return x + y
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Again we deal with the same situation as in the first example, but now the left operand is an integer, and the right one is a string. You cannot add a string to an integer - clear message in the `TypeError`. Once more, the incompatibility is the issue here, and the line of code we should jump to is the next line - line 46. (We can see a pattern here...)

Traceback Problem 3

```
Traceback (most recent call last):
  File "stack_traces.py", line 36, in run_trace
    f()
  File "stack_traces.py", line 47, in <lambda>
    run_trace(3, lambda: perform_calculation(mult, '3', '3'))
  File "stack_traces.py", line 8, in perform_calculation
    calc(x, y)
  File "stack_traces.py", line 15, in mult
    return x * y
TypeError: can't multiply sequence by non-int of type 'str'
```

Once more a similar situation, but this time not so obvious. In Python we can multiple a sequence by an integer - that's what Python tried to do. A `str` is also a sequence:

```python
from typing import Sequence
assert isinstance("G.R. Emlin", Sequence)
```

The second operand was also a string - hence the `TypeError`. Once more the incompatibility of operands caused an `Exception`. We should jump to - no suprise here - the 47th line.

Traceback Problem 4

```
Traceback (most recent call last):
  File "stack_traces.py", line 36, in run_trace
    f()
  File "stack_traces.py", line 48, in <lambda>
    run_trace(4, lambda: perform_calculation(mult, [4], [3]))
  File "stack_traces.py", line 8, in perform_calculation
    calc(x, y)
  File "stack_traces.py", line 15, in mult
    return x * y
TypeError: can't multiply sequence by non-int of type 'list'
```

Same as before, but the right operand is not a string but a list. We should jump to the 48th line.

Traceback Problem 5

```
Traceback (most recent call last):
  File "stack_traces.py", line 36, in run_trace
    f()
  File "stack_traces.py", line 49, in <lambda>
    run_trace(5, lambda: perform_calculation(innoc, '1', 3))
  File "stack_traces.py", line 8, in perform_calculation
    calc(x, y)
  File "stack_traces.py", line 22, in innoc
    spelunk()
  File "stack_traces.py", line 21, in spelunk
    raise ValueError('Invalid')
ValueError: Invalid
```

It's a tricky one. This time, we have a `ValueError` raised directly in the code by the `spelunk()` function which is called in the `innoc()` function. We should examine the code of the latter.

Given the line numbering, we can recreate the code:

```
19: def innoc(*_):
20:     def spelunk():
21:         raise ValueError('Invalid')
22:     spelunk()
```

Regardless of our recreation attempt, we should examine the 21st line and find the reason for an error.

Traceback Problem 6

```
Traceback (most recent call last):
  File "stack_traces.py", line 36, in run_trace
    f()
  File "stack_traces.py", line 50, in <lambda>
    run_trace(6, lambda: comp_calc([1, 2, 3], 1, add))
  File "stack_traces.py", line 30, in comp_calc
    return [perform_calculation(calc, x_i, y_i) for x_i, y_i in zip(x, y)]
TypeError: zip argument #2 must support iteration
```

At first sight, the complicated list comprehension on line 30 could disturb us, but it seems to be ok. The `TypeError` message is quite self-descriptive though*. The second argument, most likely on line 50 - an integer, indeed does not support iteration. We should examine this line.

*In Python 3.8, the error message would be: `'int' object is not iterable`.

Traceback Problem 7

```
Traceback (most recent call last):
  File "stack_traces.py", line 36, in run_trace
    f()
  File "stack_traces.py", line 51, in <lambda>
    run_trace(7, lambda: comp_calc([1, 2, [3]], [4, 5, 6], add))
  File "stack_traces.py", line 30, in comp_calc
    return [perform_calculation(calc, x_i, y_i) for x_i, y_i in zip(x, y)]
  File "stack_traces.py", line 30, in <listcomp>
    return [perform_calculation(calc, x_i, y_i) for x_i, y_i in zip(x, y)]
  File "stack_traces.py", line 8, in perform_calculation
    calc(x, y)
  File "stack_traces.py", line 12, in add
    return x + y
TypeError: can only concatenate list (not "int") to list
```

This time the `zip()` call is fine. However, members of iterables passed to zip are most likely the problem. Once more, the `TypeError` message is clear: an integer cannot be concatenated to list. The reason is probably the pair: `([3], 6)`. We should examine the line 51.

Traceback Problem 8

```
Traceback (most recent call last):
  File "stack_traces.py", line 36, in run_trace
    f()
  File "stack_traces.py", line 52, in <lambda>
    run_trace(8, lambda: calc_dict({'one': 1, 'two': '2'}, 'one', 'two', add))
  File "stack_traces.py", line 26, in calc_dict
    return perform_calculation(calc, d[k1], d[k2])
  File "stack_traces.py", line 8, in perform_calculation
    calc(x, y)
  File "stack_traces.py", line 12, in add
    return x + y
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

This is similar to the second example. However, now we have a mysterious `calc_dict` function. Given its name and arguments, we can assume that `1` and `'2'` are arguments passed down to the `add()` function. Once again, we should examine them by jumping to the 52nd line.

Traceback Problem 9

```
Traceback (most recent call last):
  File "stack_traces.py", line 36, in run_trace
    f()
  File "stack_traces.py", line 53, in <lambda>
    run_trace(9, lambda: calc_dict({}, 'one', 'two', add))
  File "stack_traces.py", line 26, in calc_dict
    return perform_calculation(calc, d[k1], d[k2])
KeyError: 'one'
```

Once more, given the function (`calc_dict()`) name, and arguments passed to it, we can safely assume that the reason for an error is that, indeed, an empty dictionary has no `'one'` key. We should jump to the 53rd line.

Reverse engineering

After the above analysis, given the similar structure of each problem, and growing line numbers, we can easily recreate the code that generated those tracebacks.

The result can be seen in the appendix as well as on the next page. The latter has few whitespaces less to fit on one page

```python
import sys
import traceback


def perform_calculation(calc, x, y):
    calc(x, y)


def add(x, y):
    return x + y


def mult(x, y):
    return x * y


def innoc(*_):
    def spelunk():
        raise ValueError('Invalid')
    spelunk()


def calc_dict(d, k1, k2, calc):
    return perform_calculation(calc, d[k1], d[k2])


def comp_calc(x, y, calc):
    return [perform_calculation(calc, x_i, y_i) for x_i, y_i in zip(x, y)]


def run_trace(i: int, f):
    try:
        f()
    except Exception:
        print(f'Traceback Problem {i}\n====================', file=sys.stderr)
        traceback.print_exc()
        print('\n', file=sys.stderr)

run_trace(1, lambda: perform_calculation(add, '1', 3))
run_trace(2, lambda: perform_calculation(add, 7, '3'))
run_trace(3, lambda: perform_calculation(mult, '3', '3'))
run_trace(4, lambda: perform_calculation(mult, [4], [3]))
run_trace(5, lambda: perform_calculation(innoc, '1', 3))
run_trace(6, lambda: comp_calc([1, 2, 3], 1, add))
run_trace(7, lambda: comp_calc([1, 2, [3]], [4, 5, 6], add))
run_trace(8, lambda: calc_dict({'one': 1, 'two': '2'}, 'one', 'two', add))
run_trace(9, lambda: calc_dict({}, 'one', 'two', add))
```