Lab 7 Resource Document: Stack Traces

Python prints a **stack trace** when it encounters a runtime error. These messages can seem dense, but have a lot of useful information for debugging.

Here's an example stack trace from an implementation of **word_search.py** that I wrote:

```
Traceback (most recent call last):
  File "/Users/jordan/Desktop/proj01long/word_search.py", line 112, in <module>
    main()
  File "/Users/jordan/Desktop/proj01long/word_search.py", line 105, in main
    found_words = find_words_in_grid(letter_grid, word_list)
                  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/Users/jordan/Desktop/proj01long/word_search.py", line 60, in
find_words_in_grid
    possible_lines.append(get_diagonal(grid, i))
                          ^^^^^^^^^^^^^^^^^^^^^
  File "/Users/jordan/Desktop/proj01long/word_search.py", line 28, in get_diagonal
    retlist.append(grid[pos[0]][pos[1]])
                   ~~~~^^^^^^^^
IndexError: list index out of range
```

Note the following features:

1. **Traceback (most recent call last):** It shows the earliest function call at the top, in this case **main()**.
2. **File "..." line X, in <function>:** The rest of the trace shows which functions called each other leading up to the error. **main()** called **get_diagonal(grid, i)**, which tried to append to **retlist**. It shows which line and where the error occurred.
3. **IndexError: list index out of range:** We can see that the **IndexError** occurred when we attempted to access **grid[pos[0]]**.

The stack trace allows you to follow what happened from the start of your program until the error occurred. Sometimes, the actual error you need to fix will be at the bottom of the trace—the issue in the above example was a bad loop condition inside **get_diagonal**. Other times, a function in the middle of the trace may cause an error down the line. It's often best to work your way bottom to top

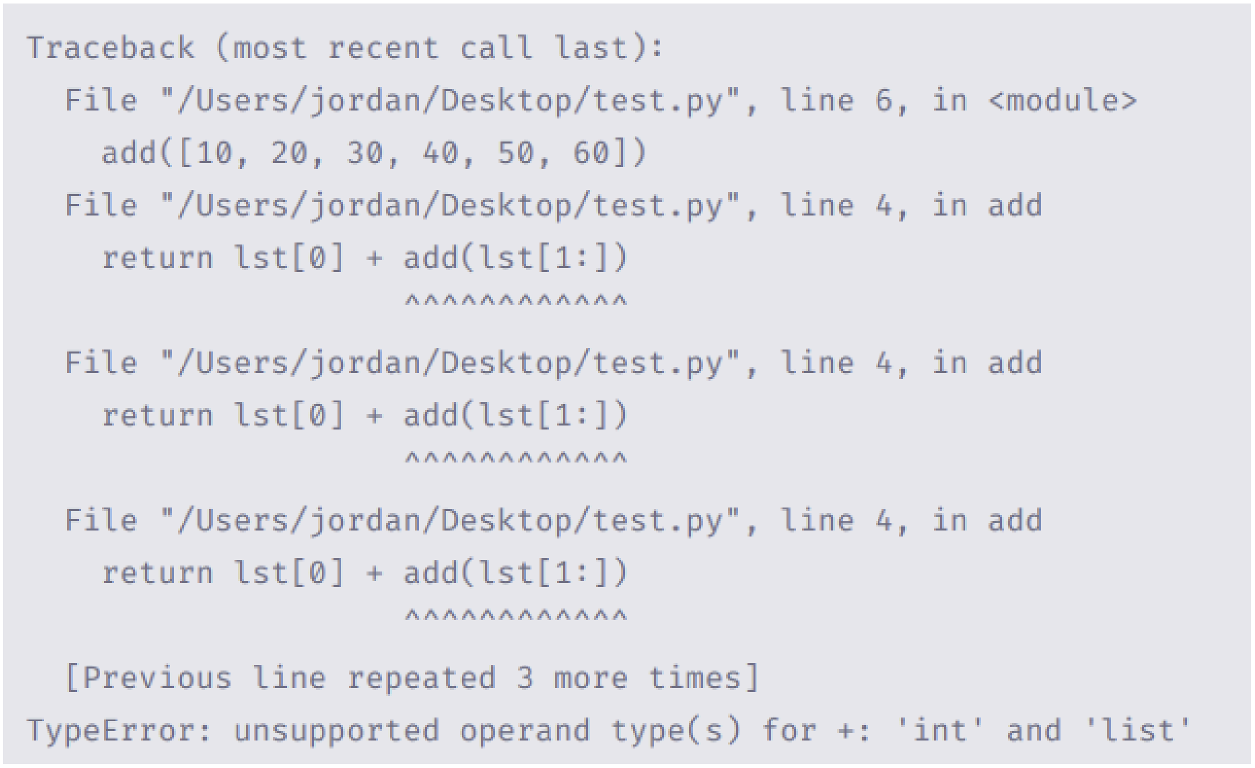
Recursion

Here's an example of a recursive function with a common error:

```python
def add(lst):
    if lst == []:
        return lst
    return lst[0] + add(lst[1:])

add([10, 20, 30, 40, 50, 60])
```

And here's the traceback when this code is executed:



```
Traceback (most recent call last):
  File "/Users/jordan/Desktop/test.py", line 6, in <module>
    add([10, 20, 30, 40, 50, 60])
  File "/Users/jordan/Desktop/test.py", line 4, in add
    return lst[0] + add(lst[1:])
                    ^^^^^^^^^^^^

  File "/Users/jordan/Desktop/test.py", line 4, in add
    return lst[0] + add(lst[1:])
                    ^^^^^^^^^^^^

  File "/Users/jordan/Desktop/test.py", line 4, in add
    return lst[0] + add(lst[1:])
                    ^^^^^^^^^^^^

  [Previous line repeated 3 more times]
TypeError: unsupported operand type(s) for +: 'int' and 'list'
```

With recursive functions, you get a repetitive trace from the function calling itself. What matters here is that bottom line showing what the error refers to: we're trying to add together an **int** and a **list** when we do that final recursive call. Of course, to fix this, we have to ensure that our base case returns the same type as our recursive case.