Part 5: Debugging

Bookmark this page

Part 5: Debugging

We said that logging and debugging are a step up from the print statements that many new programmers use to debug their code.

The first half of this lesson presented logging as a direct evolution to print statement debugging: a logging statement is like a print statement that can be hidden, or have extra information attached to it, or can be sent to somewhere other than the console.

Debugging your code with an interactive debugger is another thing entirely: although both practices help you answer the same question of what's going in in your code, interactive debugging is not at all like inserting print statements.

Interactive debugging allows you to run the Python interpretter line-by-line through your code, pausing to print out the values of particular variables, or to evaluate other statements inside of the interpretter. And it includes tools that can help you "zoom through" the execution of many statements to get right to trouble-raising conditions in your code.

Basic Debugging Commands

Let's begin by understanding the basic commands of the interactive debugger. We'll begin by debugging the file simple.py in the debugging exercises code repository:

```
# simple.py
def my_fun():
    for i in range(1, 500):
         123/ (50 - i)

if __name__ == '__main__':
    my_fun()
```

Try running the script. You probably expected to receive a ZeroDivisionError. Let's use this code to begin exploring the Python interactive debugger.

Next, we will look at really_simple.py.

Video 1 of 6: Navigating in the Debugger

Start of transcript. Skip to the end.

[MUSIC PLAYING]

The goal of this video is to explore the basic commands

of the interactive Python debugger, especially navigating

through the execution of a script.

So here is reallysimple.py, and when we run it, we get a division by 0 error.

And that's no surprise by looking at the code

Video

Download video file

Transcripts

Download SubRip (.srt) file

Download video file

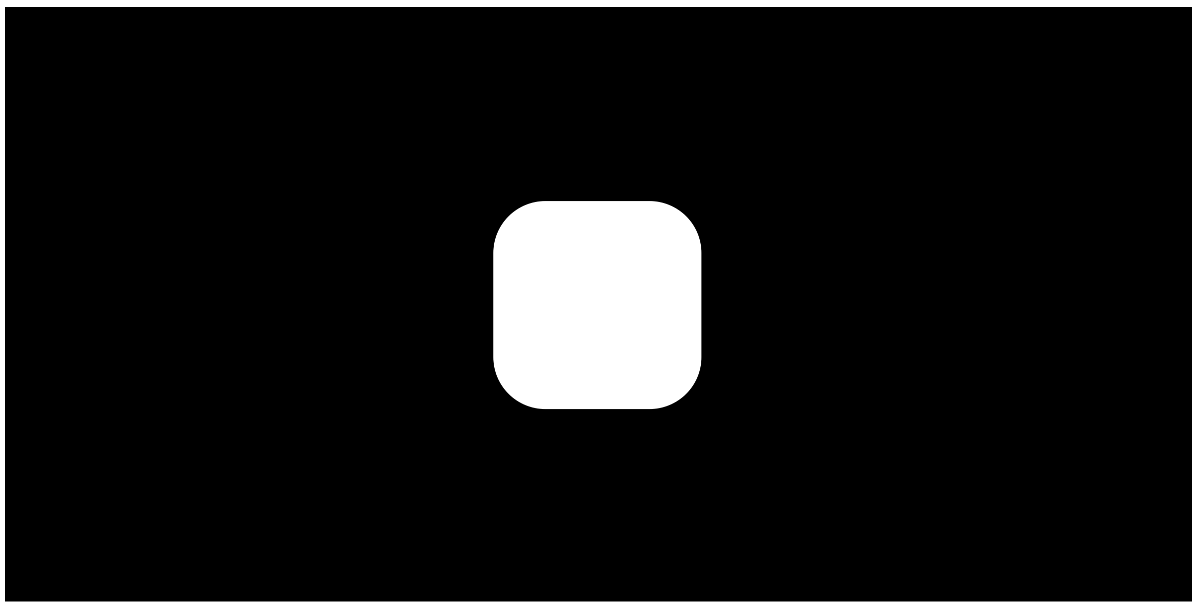
Download SubRip (.srt) file
Download Text (.txt) file

Breakpoints

It could take a lot of 's' and 'n' commands to get to that ZeroDivisionError condition in simple.py! Breakpoints and conditions allow you to "zoom through" the execution of your code, pausing the interpretter when a certain condition on a certain line of code is met.

Next, we'll look at simple_example.py.

Video 2 of 6: Setting Conditions and Breakpoints



Start of transcript. Skip to the end.

You've learned how to navigate through the execution of code

in the debugger using the s and n commands,

But sometimes, there might be a lot of code

or a lot of execution between the beginning of a script and an error

condition that you're interested in.

Here's simpleexample.py.

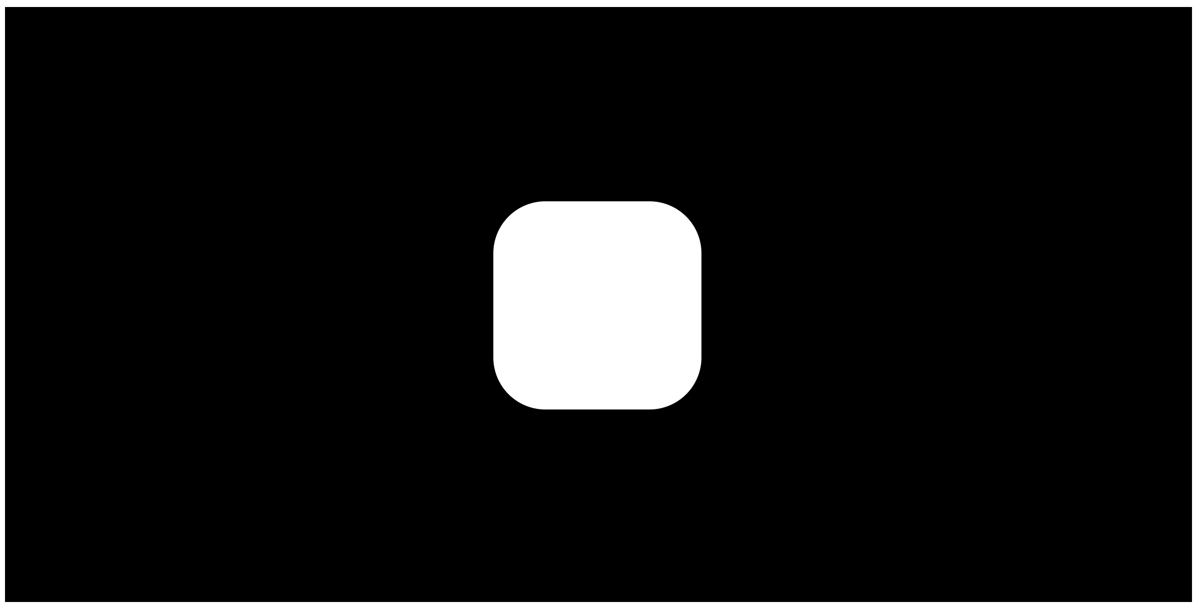
Video
Download video file

Transcripts
Download SubRip (.srt) file
Download Text (.txt) file

Complicated Example and Exercise

Here's an exercise where the error in our code is not entirely obvious, complicated_example.py.

Video 3 of 6: Debugging Exercise



Start of transcript. Skip to the end.

[MUSIC PLAYING]

Next let's do an exercise with the file ComplicatedExample.py.

It happens to me all the time.

I meet with a client and they say, we've got this i thing,

and it varies between 2 and 1,000 by steps of 5.

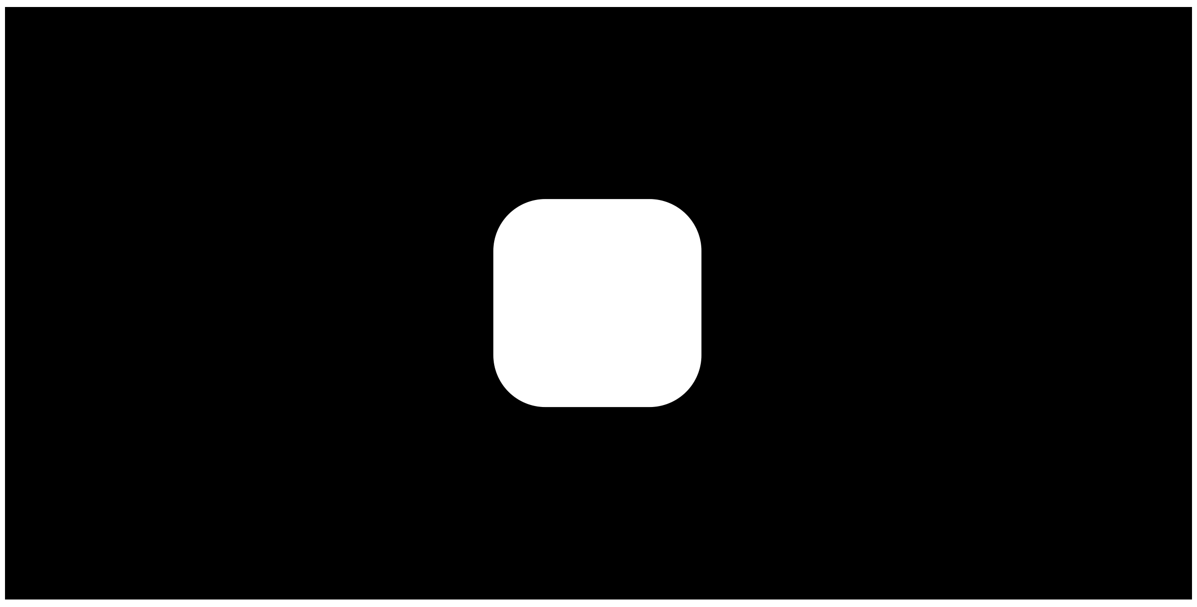
And this j that varies between 3 and
1,000 by steps of 7

Video
Download video file

Transcripts
Download SubRip (.srt) file
Download Text (.txt) file

Take some time to try to figure out what values of *i*, *j*, and *k* give rise to the zero division error. Focus on trying to create a breakpoint condition for line 19 that will be met if the interpretter is *about* to divide by zero.

Video 4 of 6: Debugging Exercise Solved



Start of transcript. Skip to the end.

I'm trying to answer the question, what values of I, J, and K

give rise to the 0 division error in complicated example.pi.

Here's the error.

And I'm going to open this in the debugger and do some exploring.

And right now I could end and S through to get to the-- into the loop

Video
Download video file

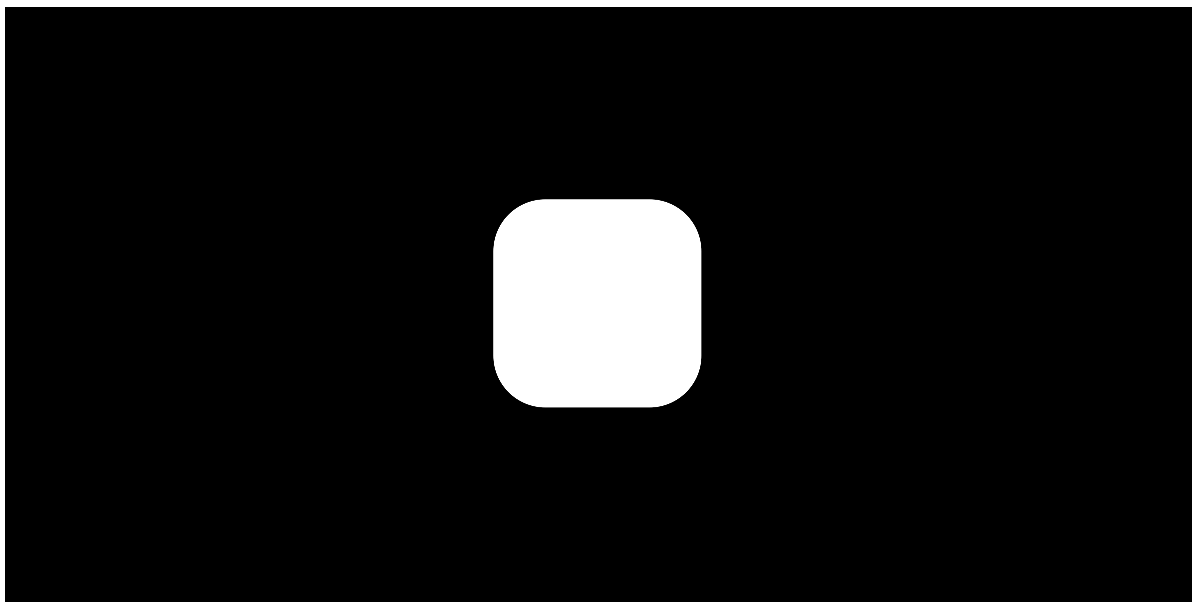
Transcripts
Download SubRip (.srt) file
Download Text (.txt) file

Recursion Error Exercise

Here's an exercise that finally does not involve a ZeroDivisionError! Instead, we'll be investigating a RecursionError.

For the lesson activity, you'll be required to copy and your debugger output from this recursion exercise and paste it into the activity submission text box. Before beginning this video, visit the lesson activity to make sure that you understand what will be required.

Video 5 of 6: Exercise in Recursion



Start of transcript. Skip to the end.

[MUSIC PLAYING]

This debugging puzzle involves a recursive function,

and a recursive function is just a function that might call itself in order to perform its duty.

So let's take a look at the function in the file recursive.py.

Here's the function myfun.

Video
Download video file

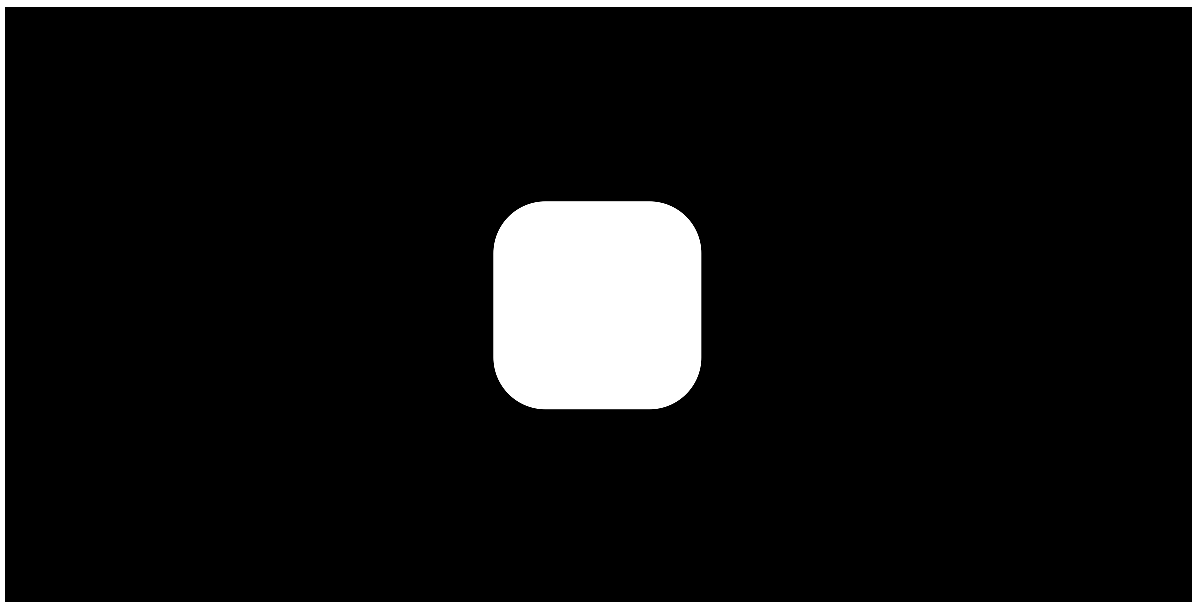
Transcripts
Download SubRip (.srt) file
Download Text (.txt) file

Use the interactive debugger to analyze the error in our program. In a couple of sentences, describe our error in the following terms:

- What is wrong with our logic?
- Why doesn't the function stop calling itself?
- What's happening to the value of 'n' as the function gets deeper and deeper into recursion?

Once you're satisfied with your answer, see the next video:

Video 6 of 6: Exercise in Recursion Solved



Start of transcript. Skip to the end.

Let's answer the question, what's going wrong in our recursion.pi script.

So for demonstration purposes, I'm going to start off with an actual power of 2.

We're going to debug our script with a value of 16

to see how the code executes in this happy case.

I'll run Python minus m PDB

Video
Download video file

Transcripts
Download SubRip (.srt) file
Download Text (.txt) file

Conclusion