LECTURE 11

THE PYTHON DEBUGGER

MCS 275 Spring 2021
Emily Dumas

LECTURE 11: THE PYTHON DEBUGGER

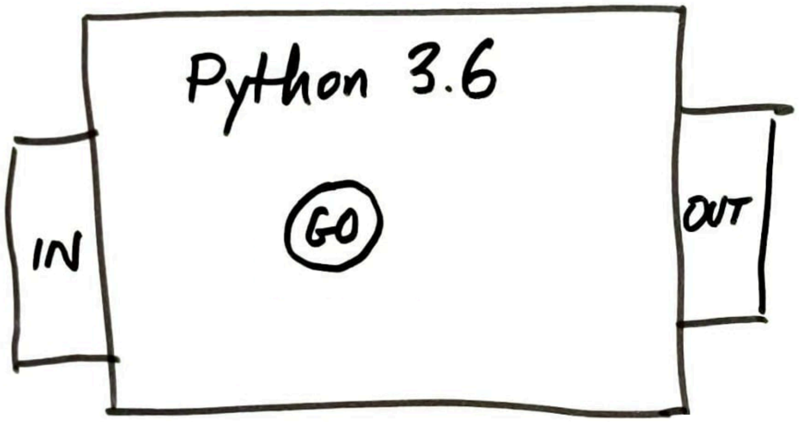
Course bulletins:

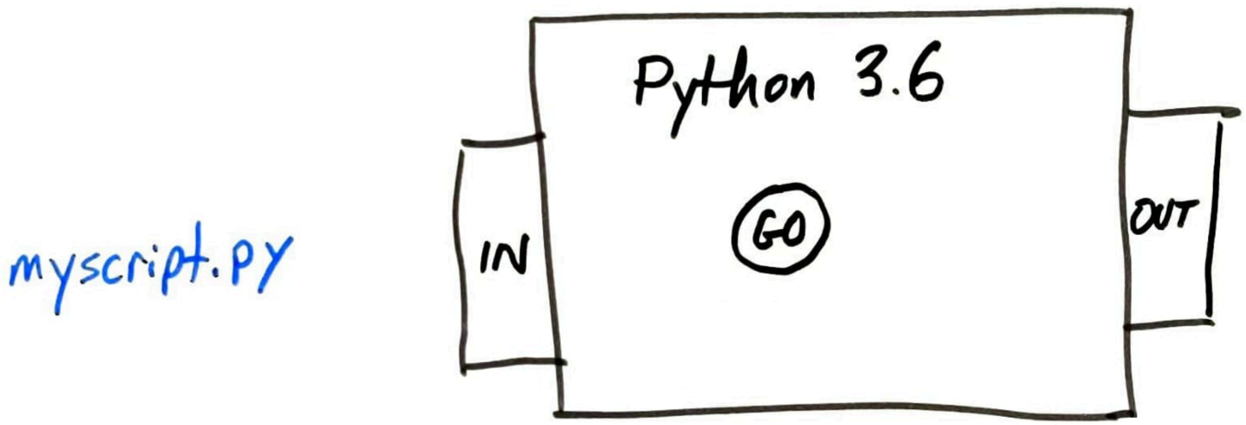
- Project 1 due today at 6pm CST.
- Next week we begin a more theoretical unit (recursion). Check course web page for supplemental reading suggestions.

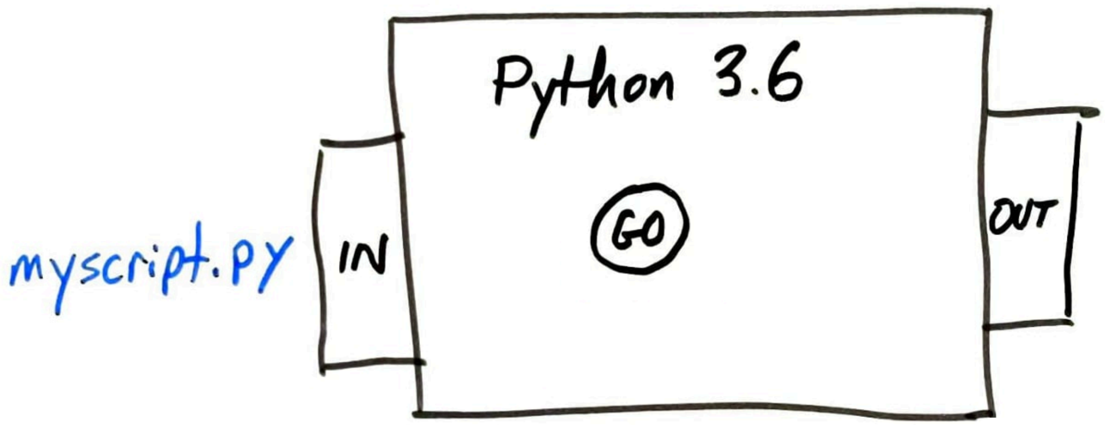
THE IDEA OF A DEBUGGER

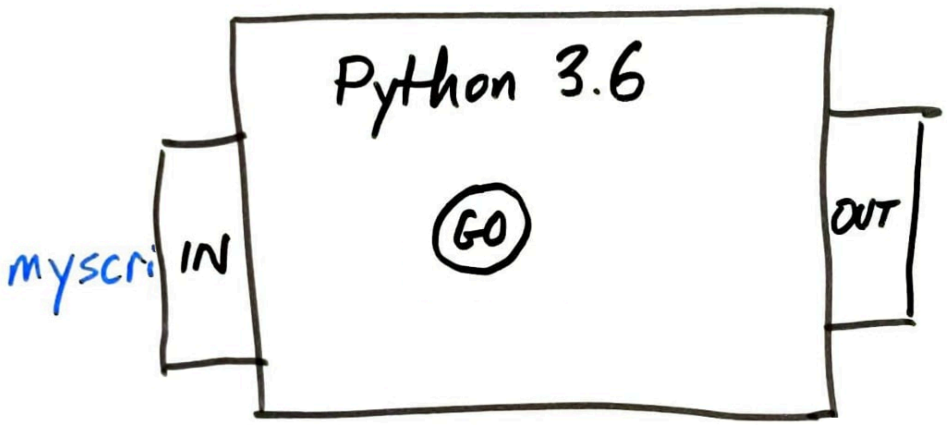
Suppose a Python program has a bug.

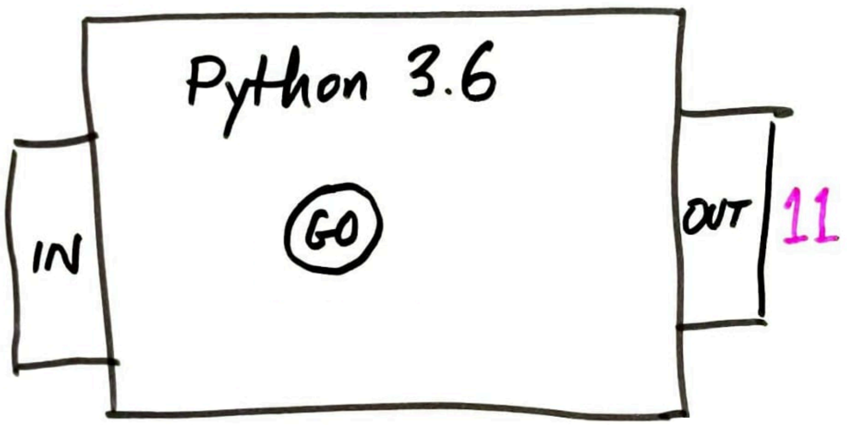
Wouldn't it be nice if you could run the program slowly, monitoring values of variables along the way?

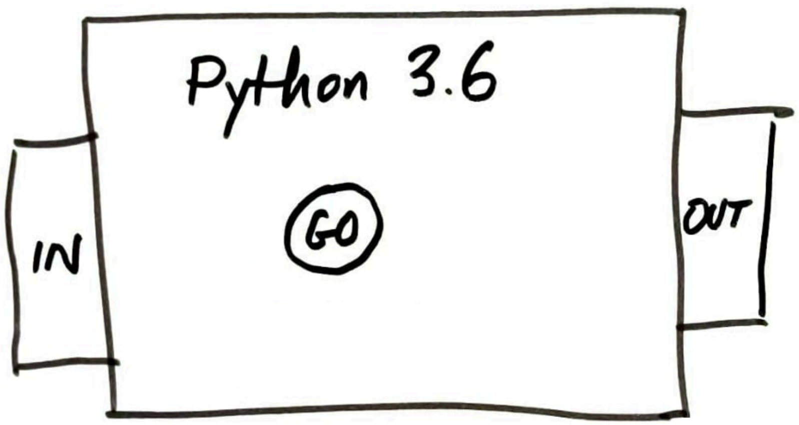

Python 3.6
IN
GO
OUT


myscript.py
Python 3.6
IN
GO
OUT


Python 3.6
myscript.py
IN
GO
OUT

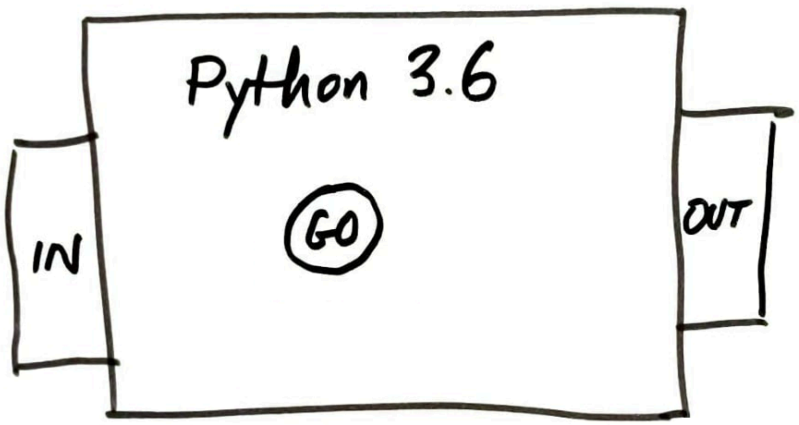

Python 3.6
GO
IN
OUT
myscri

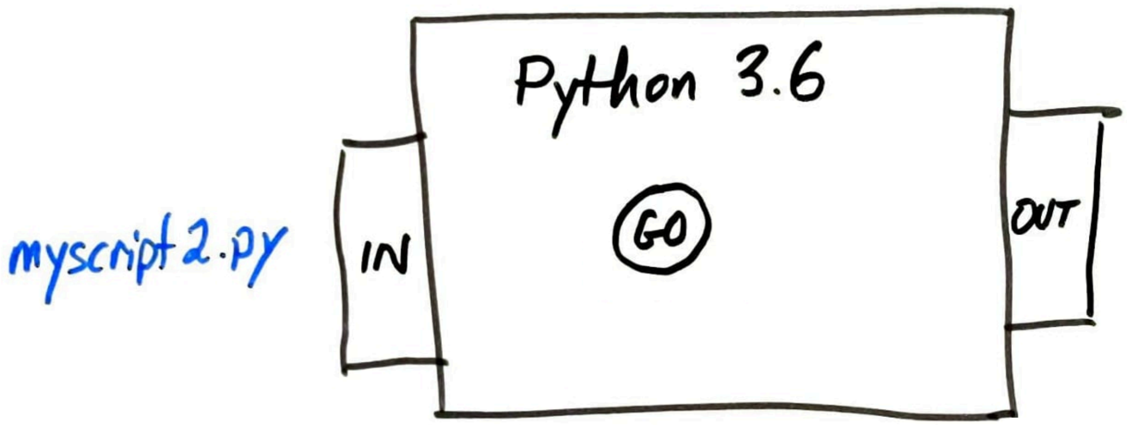

Python 3.6
IN
GO
OUT
11

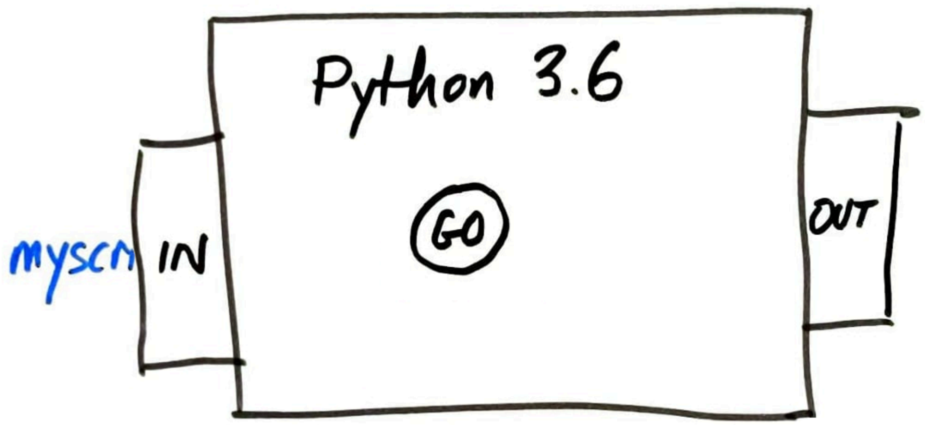


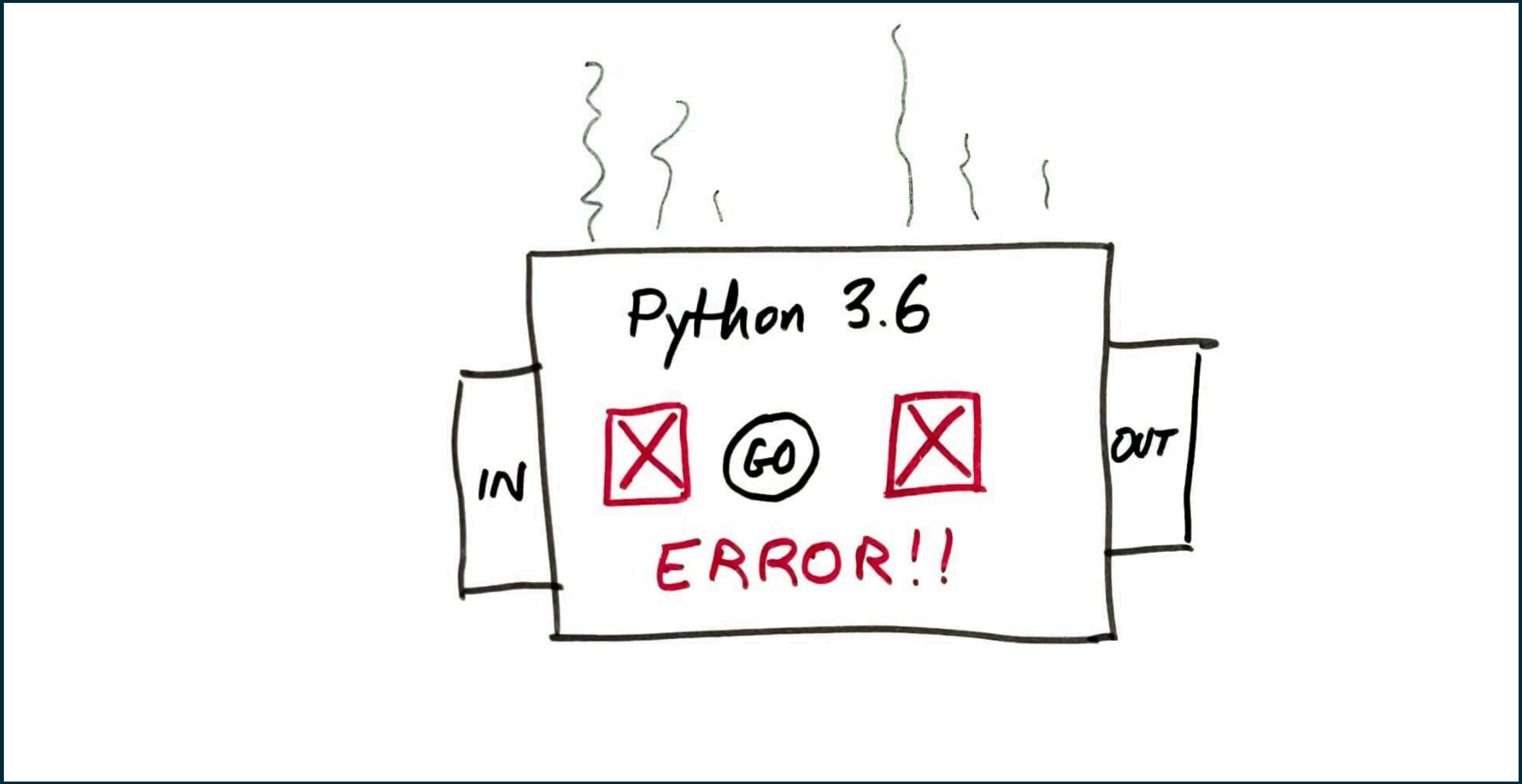
11

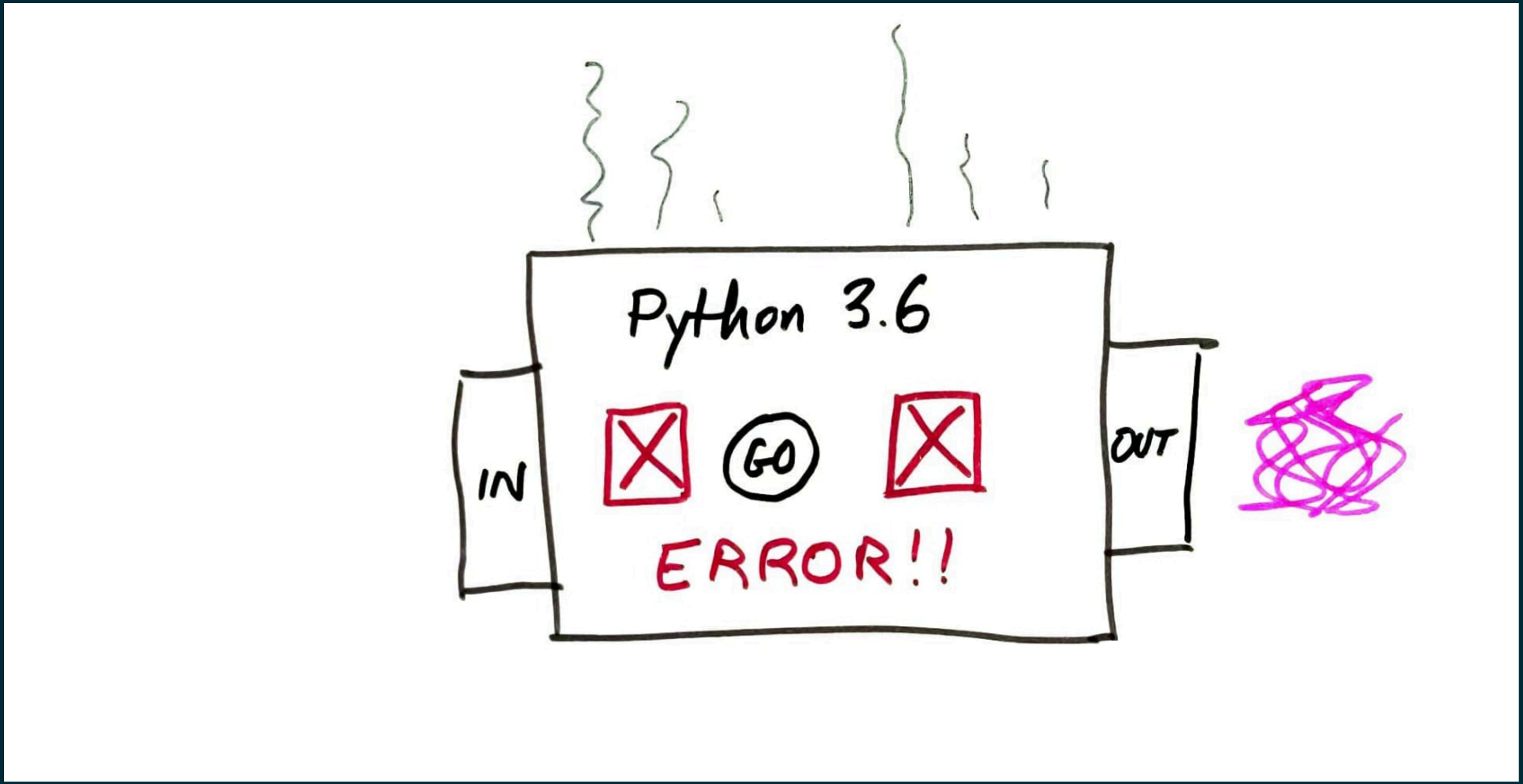
myscript2.py

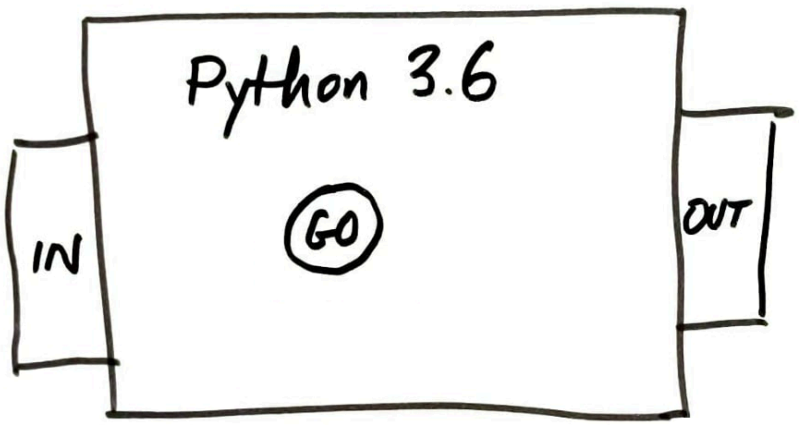


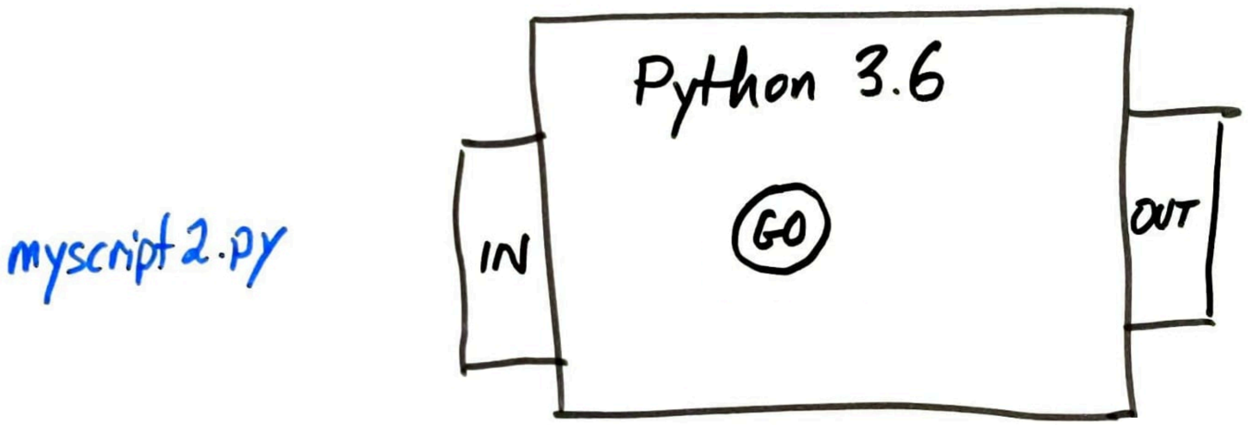

myscript2.py
IN
Python 3.6
GO
OUT

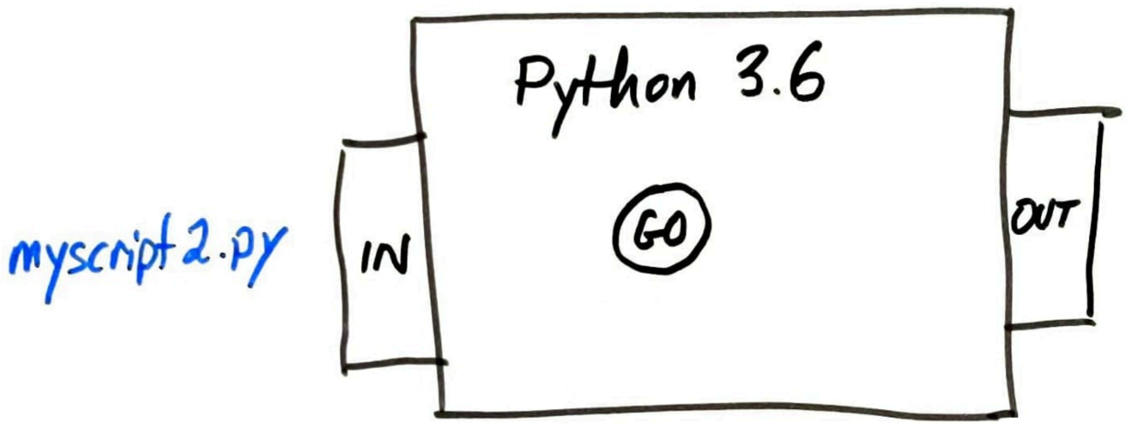

Python 3.6
GO
myscri
IN
OUT

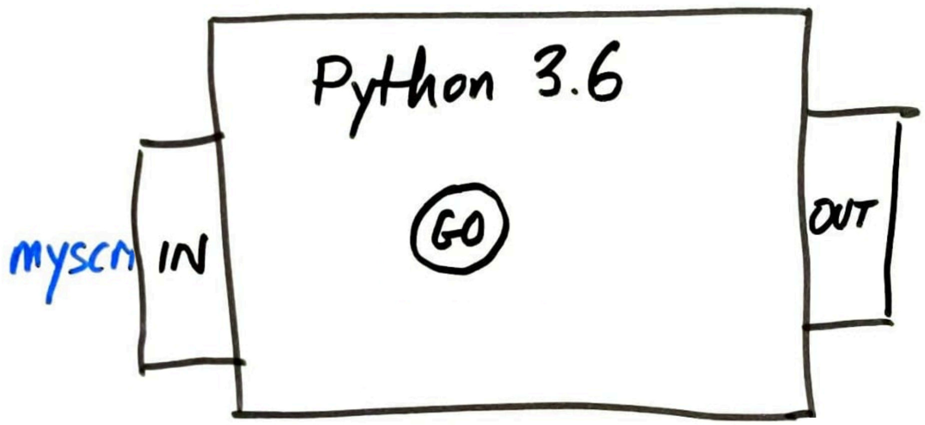

Python 3.6
IN
GO
OUT
ERROR!!

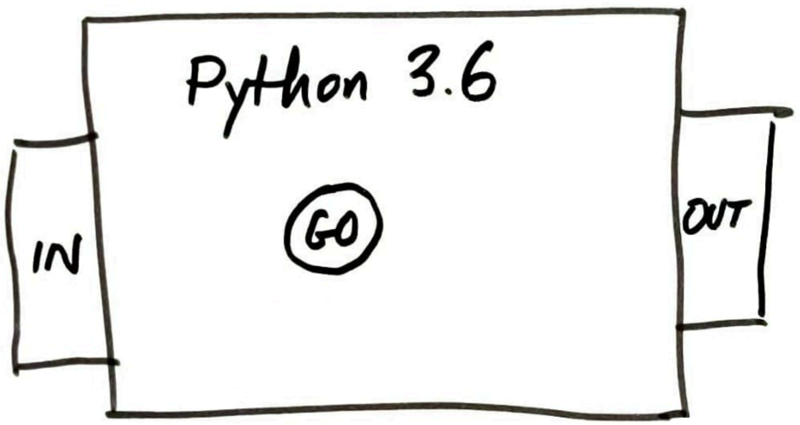

Python 3.6
IN
GO
OUT
ERROR!!

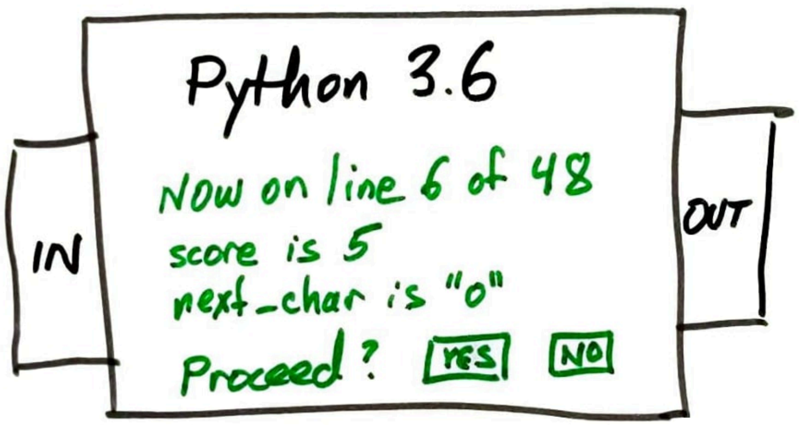

Python 3.6
IN
GO
OUT


Python 3.6
myscript2.py
IN
GO
OUT


Python 3.6
GO
IN
OUT
myscript2.py


Python 3.6
GO
myscri
IN
OUT


Python 3.6
IN
GO
OUT


Python 3.6
IN
OUT
Now on line 6 of 48
score is 5
next_char is "o"
Proceed? YES NO

PDB

The built-in Python debugger, called `pdb`, makes this possible. Key features:

- Single-step through a program
- Inspect values of variables
- Run normally until a certain line is reached
- Analyze an exception that is about to end the program.

RUNNING PDB

```
python -m pdb myprogram.py alpha beta 3
```

Runs `myprogram.py` with command line arguments `["alpha","beta",3]`, but in the debugger.

The program starts in a **paused state**, and a prompt is shown where we can enter commands (to resume, run a single line, show values of variables, etc.)

PDB BASICS

Running the program:

- **c** or **continue** -- Start or continue running the program, i.e. "unpause".
- **s** or **step** -- Start execution but stop as soon as possible. If a function is called, move to the first line of that function.
- **n** or **next** -- Start execution and stop when the next line of the current function is reached. (If the current line calls other functions, wait for them to return.)
- **r** or **return** -- Start execution and stop when the current function returns.

Inspecting the situation:

- **l** or **list** -- Show a passage of source code that includes the current line.
- **ll** -- Show the entire contents of the file containing the current line.
- **`pp`** `EXPR` -- Evaluate `EXPR` and display the result nicely ("pretty print")
- **`display`** `EXPR` -- Every time execution is paused, show the value of `EXPR` if it has changed.

BREAKPOINTS

Rather than single-stepping, it is often helpful to keep running until a certain part of the code is reached.

A place where execution is supposed to stop and return control to the debugger is a **breakpoint**.

- **`b`** `FILE:LINE_NUM` -- Set breakpoint by line.
- **`b`** `FUNCTION_NAME` -- Set a breakpoint by function name.
- **`cl`** -- Clear all breakpoints.

POST MORTEM

If an uncaught exception occurs when a program is running in `pdb`, the debugger pauses at the moment of the exception to let you investigate.

This is called a "post mortem" (after death) investigation of the program. You can't continue or step, but you can examine the values of variables, etc.

MOVING AROUND THE TRACEBACK

What if `f()` calls `g()`, and you are paused inside `g` but want to know the value of a local variable of `f`?

- **`u`** or **`up`** -- Move one step up the traceback, to the function which called this one.
- **`d`** or **`down`** -- Move one step down the traceback, to the function which called this one.
- **`w`** or **`where`** -- Show the current traceback.

REFERENCES

- pdb is not discussed in any depth in the optional texts.
- The official pdb documentation
- This Tutorial on the Python debugger by Lisa Tagliaferri at DigitalOcean is nice.

REVISION HISTORY

- 2021-02-07 Add "where" command
- 2021-02-05 Initial publication