6 Debugging

Key terms: `pdb` breakpoint
Reading: Lisa Tagliaferri's How To Use the Python Debugger[16]
Exercise: Write a program with an infinite loop and use pdb to stop it after 1M iterations.

If we regard a program as a state machine, we can inquire about its state at any time. If a program is manifesting some surprising behavior, we might like to see what state it was in before the misbehavior starts; if a program ends prematurely, we might like to see what state it was in before exiting. You are already familiar with checking the content of objects because this is the basis of the quick-and-dirty means of debugging with `print`. Unlike interactive debugging, `print` debugging requires a code change for every different kind of analysis.

According to the software engineering reference book "Code Complete"[17], the Scientific Method of Debugging is to:

1. Reproduce the error and maybe simplify the error case.
2. Gather data that produces the defect.
3. Analyze the data and form a hypothesis about the source of the error.
4. Determine how to prove or disprove it, either through inspection or execution.
5. Prove or disprove it.
6. Fix the defect.
7. Test the fix.

`pdb`[18] is Python's interactive debugger, somewhat modeled on `gdb`. This document discusses `pdb`; also note the debuggers in `PyDev` (`Eclipse`), `IDLE`, `PyCharm`, and `PuDB`.

In `pdb`, debugger commands can be mixed with plain Python ones; when a command does not appear to be for the debugger, or if it's prefaced with !, it is executed as Python. Debug execution is slower than normal execution (which can be another reason to find a reduced test case).

```
[yourname@hills] $ python3 -m pdb transcend.py
> /home/yourname/transcend.py(3)<module>()
-> import sys
(Pdb)
```

`pdb` lets us run and stop the program as needed, to monitor its operations. By inquiring about the contents of objects we can track the state of the program. Even programs that have already died can be inspected. `s` steps forward, including into functions; `n` proceeds to the next line. `l` shows the current location in the program listing. `w`, "where", shows a stack trace (try this with a recursive program). `help command` shows documentation for using for `command`.

```
> /home/yourname/bad.py(13)<module>()
-> store [word] = True
(Pdb) type(store),type(word)
(<class 'list'>, <class 'str'>)
```

A breakpoint is where you'd like execution to halt for a state inspection. Literally, execution continues until the breakpoint, where interactive control returns to the user of the debugger. Breaks can be set at certain line numbers, line numbers in certain files, certain functions, and also subject to certain conditions. Each one gets a number. Breakpoints can even trigger custom commands. `b` sets a breakpoint or shows breakpoints; `c` continues to the next one; `cl` clears them.

By comparison, debugging functional code means validating the data at different levels of nesting. A passthrough function which returns the same thing it is passed after logging can do this.

[16] https://www.digitalocean.com/community/tutorials/how-to-use-the-python-debugger
[17] https://en.wikipedia.org/wiki/Code_Complete
[18] https://docs.python.org/3/library/pdb.html