

Debugging without print()

...

An Introduction to PDB

Matt Curcio & Emily Charles
March 9, 2023

Bugs or Moths?

1947 Harvard Mark II Team on a calculator program

9/9

0800 Anchan started

1000 " stopped - anchan ✓

1300 (032) MP-MC { 1.2700 9.037 847 025
2.130476415 (2) 4.615925059(-2)

(033) PRO 2 2.130476415

conv 2.130676415


Relays 6-2 in 033 failed special speed test
in relay " 11.000 test.

Relays changed

1100 Started Cosine Tape (Sine check)

1525 Started Multi-Adder Test.

1545



Relay #70 Panel F
(moth) in relay.

First actual case of bug being found.

1650 Anchan started.

1700 closed down.

Relay
2142
Relay 1170

What, when, where, why pdb?

- Python **de**bbugger (Pdb) comes 'Built-in'
- No GUI's? Use CLI
- More info. than: `print(f'a = {a}')`
- Step thru a program
- Check ANY Variables
- No need to remove 'print' statements later

Alternatives

- 'print' is very simple
- Use Logging ala Glenn
- ipdb, pudb, pdb++, PyCharm, IDE's
- Even more?

<https://wiki.python.org/moin/PythonDebuggingTools>

Ready, Steady, Go!

Option 1: `$ python3 -m pdb title.py` `# -m(module)`

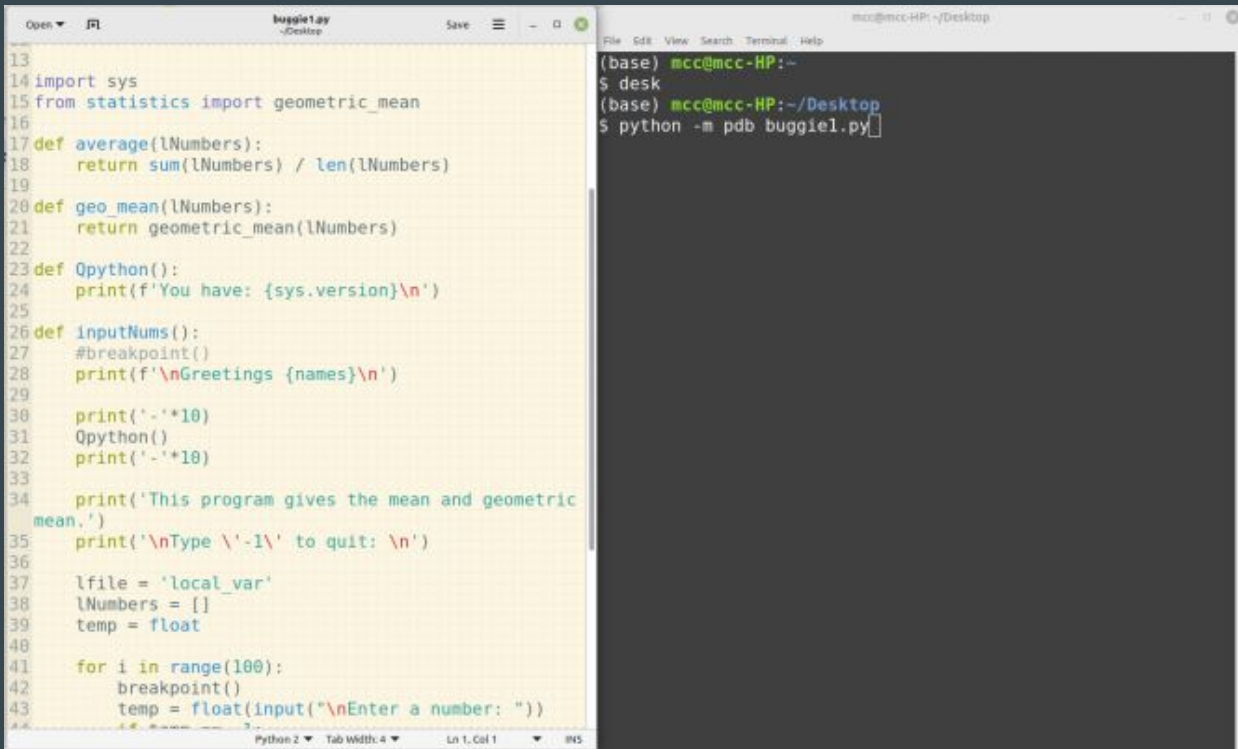
Option 2: Set breakpoints in code

`>3.7 breakpoint()` `<3.7 import pdb; pdb.set_trace()`

Option 3: `__import__('pdb').set_trace()`

One Possible Workflow

Step 1 Open terminal & your code in a text editor



The screenshot displays a workflow for running a Python script. On the left, a text editor window titled 'buggie1.py' shows a Python program. The code defines functions for calculating the average and geometric mean of a list of numbers, and includes a loop for user input. On the right, a terminal window shows the execution of the script using the 'python' command, with the output of the program visible.

```
13
14 import sys
15 from statistics import geometric_mean
16
17 def average(lNumbers):
18     return sum(lNumbers) / len(lNumbers)
19
20 def geo_mean(lNumbers):
21     return geometric_mean(lNumbers)
22
23 def Opython():
24     print(f'You have: {sys.version}\n')
25
26 def inputNums():
27     #breakpoint()
28     print(f'\nGreetings {names}\n')
29
30     print('-'*10)
31     Opython()
32     print('-'*10)
33
34     print('This program gives the mean and geometric
mean.')
```

Help

- @ Pdb: 'h' or 'help' <command>
- Pdb has NO man page!?!
- <https://docs.python.org/3/library/pdb.html>
- [Matt's PDF](#)
- Google: 'python pdb' NOT the Protein DataBase (PDB)

Pdb Interpreter Acts like Python

```
mcc@mcc-HP: ~/Desktop
File Edit View Search Terminal Help
(base) mcc@mcc-HP:~
$ desk
(base) mcc@mcc-HP:~/Desktop
$ python -m pdb buggie1.py
> /home/mcc/Desktop/buggie1.py(1)<module>()
-> ""
(Pdb) h

Documented commands (type help <topic>):
=====
EOF      c          d          h          list       q          rv         undisplay
a         cl         debug      help       ll         quit       s          unt
alias    clear      disable    ignore     longlist   r          source     until
args     commands  display    interact   n          restart    step       up
b         condition down        j          next       return     tbreak     w
break    cont       enable     jump       p          retval     u          whatis
bt        continue  exit       l          pp         run        unalias    where

Miscellaneous help topics:
=====
exec  pdb

(Pdb)
```


One Possible Workflow

Step 2 cli: `python -m pdb title.py` # Start pdb

Step 3 Find error(s) via:

Pdb: Use `c(ontinue)`, `n(ext)`, or `s(tep)`

Step 4 Edit code in your text editor

Step 5 Pdb: `r(estart)` your code & find next error(s)

Step 6 Repeat

What You Cannot Do

- You cannot find logical errors

For example:

- Calculation errors, $F=m/a$, instead of $F=ma$
- Deleting rows instead of columns
- Counting from 1 instead of 0, etc.

What You Can Do

SYNTAX ERRORS

- Closing parenthesis:
- Using invalid characters:
- Missing Colons:
- Incorrect Indentation

```
print("Hello World"
```

```
a = 2 x 5; b = 10 ~ 4
```

```
if a == 10; or if a == 6
```

What You Can Do

RUNTIME ERRORS

ZeroDivisionError:

```
c = a/0
```

TypeError:

```
a = '10'; b = 5; c = a + b
```

NameError:

```
print(x), name 'x' is not defined
```

IndexError:

```
d = range(10); print(d[10])
```

Navigation within the Pdb interpreter

- `l(ist)` list 11 lines surrounding
- `n(ext)` execute the current line
- `s(tep)` step thru function from current line
- `c(ontinue)` execute until next breakpoint
- `restart` restart your program from the top

Interacting with Code

- `p <name>` print value of variable
- `!<expr>` execute the expression
- `run [args]` restart the debugger
- `arguments [args]`
- `q(uit)` exit the debugger

NOTE: this acts like a python interpreter

Now for a Live Demo

**Don't Debug,
De'moth' Python!**

Other Resources

- pdb commands explained:
<https://www.codementor.io/@stevek/advanced-python-debugging-with-pdb-g56gvmpfa>
- Ned's pdbrc:
<https://kylekizirian.github.io/ned-batchelders-updated-pdbrc.html>