Debugging

Kim Brugger

“Debugging is twice as hard as writing the code in the first place.

Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.”

-Brian W. Kernighan

Outline

Introduction

The python debugger (PDB)

Hands on

Reflection

Introduction: How 99% of debugging happens

1) Print values and/or states.
2) Learn to process text speeding across the screen
3) Run till exit
    a) Wait for a crash
    b) Wait for an unusual state then exit/stack dump
    c) Run till you have convinced yourself the code work as expected.

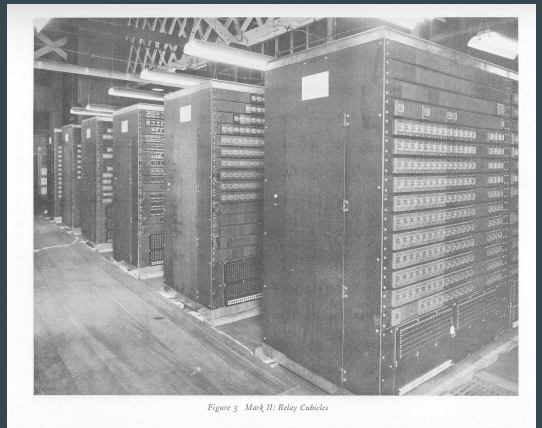
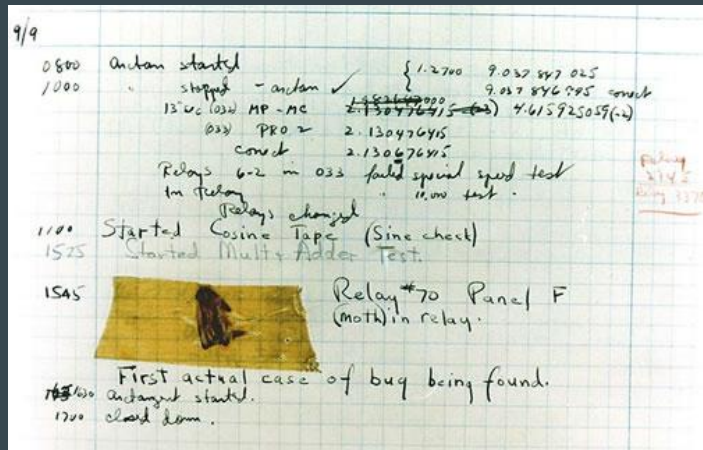


Figure 5 Mark II: Relay Cubicles



PBD: Debugging with the python debugger

```
# run script in debugging mode
python3 -m pdb app.py

# Modify the code to break on a line with a known problem
import pdb

#set breakpoint
pdb.set_trace()
X = [ x + 2 for x in range(0, 100)]
```

PDB: commands

p Print the value of an expression. (pp is pretty print)
s/n Next line or step into a function call
c Continue execution and only stop when a breakpoint is encountered.
l List source code for the current file.
ll List the whole source code for the current function or frame.
b With no arguments, list all breaks. With a line number, set a breakpoint
w Print a stack trace, with the most recent frame at the bottom.
p/d Up/Down call on the call stack

h Help, shows a list of available commands.
h <cmd> Show help for a command
q Quit the debugger and exit.

PDB: Show and Tell

Hands On: example (~30 min)

./random_list_stats.py:

INFO: Changed list length to: 20
List length: 20
Lowest: 0, highest: 10
Mean: 4
Median: 5
Mode: [0, 4, 5, 10] (count: 3)

If you change the list length (set_list_length()) it no longer generates the right list length, what is wrong, any why?

PDB: commands

p Print the value of an expression. (pp is pretty print)
s/n next line or step into a function call
c Continue execution and only stop when a breakpoint is encountered.
l List source code for the current file.
ll List the whole source code for the current function or frame.
b With no arguments, list all breaks. With a line number, set a breakpoint
w Print a stack trace, with the most recent frame at the bottom.
p/d Up/Down call on the call stack

h Help, shows a list of available commands.
h <cmd> Show help for a command
q Quit the debugger and exit.

Debugging: reflection

How did it go?

What is your normal practice?

More useful than printing variables and states?

What is a useful log message?