Introduction to Python

Debugging Python Programs

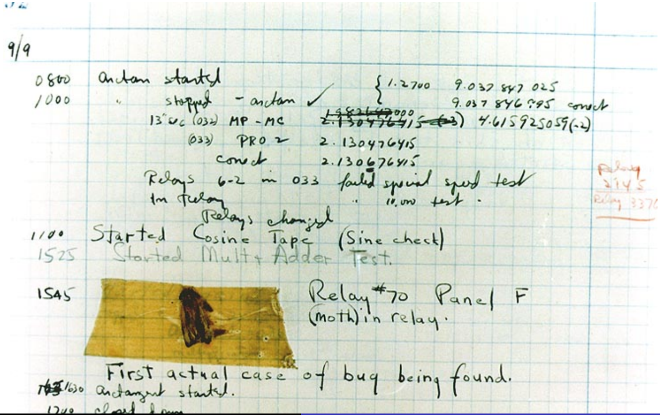
4th November 2016

What Is Debugging?

- A bug is a problem with a program - either it doesn't run or it doesn't do what you want it to do.

What Is Debugging?

- A bug is a problem with a program - either it doesn't run or it doesn't do what you want it to do.
- Named for Admiral Grace Hopper who "debugged" a moth in a Mark II in the 1940s.



9/9

0800 Antan started

1000 " stopped - antan ✓ {1.2700 9.037 847 025

9.037 846 995 conct

13" uc (032) MP - MC ~~1.982147000~~ ~~2.130476415~~ (-3) 4.615925059(-2)

(033) PRO 2 2.130476415

conct 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 Mult+ Adder Test.

1545 Relay #70 Panel F

(moth) in relay.

First actual case of bug being found.

~~1600~~ 1630 Antangent started.

[illegible]

Relay 2145
Relay 3370

What Is Debugging?

- A bug is a problem with a program - either it doesn't run or it doesn't do what you want it to do.
- Named for Admiral Grace Hopper who "debugged" a moth in a Mark II in the 1940s.
- Many different types of bugs - memory, flow control, using variables out of scope etc.

What Is Debugging?

- A bug is a problem with a program - either it doesn't run or it doesn't do what you want it to do.
- Named for Admiral Grace Hopper who "debugged" a moth in a Mark II in the 1940s.
- Many different types of bugs - memory, flow control, using variables out of scope etc.
- An **error** is behaviour in a program we don't want.

What Is Debugging?

- A bug is a problem with a program - either it doesn't run or it doesn't do what you want it to do.
- Named for Admiral Grace Hopper who "debugged" a moth in a Mark II in the 1940s.
- Many different types of bugs - memory, flow control, using variables out of scope etc.
- An **error** is behaviour in a program we don't want.
- The aim of debugging is to **diagnose** the bug that causes the error.

The Devil's Guide To Debugging

Stolen from Steve McConnell's Code Complete

1. Scatter output statements everywhere

The Devil's Guide To Debugging

Stolen from Steve McConnell's Code Complete

1. Scatter output statements everywhere
2. Debug the program into existence

The Devil's Guide To Debugging

Stolen from Steve McConnell's Code Complete

1. Scatter output statements everywhere
2. Debug the program into existence
3. Never back up earlier versions

The Devil's Guide To Debugging

Stolen from Steve McConnell's Code Complete

1. Scatter output statements everywhere
2. Debug the program into existence
3. Never back up earlier versions
4. Don't bother understanding what the program should do

The Devil's Guide To Debugging

Stolen from Steve McConnell's Code Complete

1. Scatter output statements everywhere
2. Debug the program into existence
3. Never back up earlier versions
4. Don't bother understanding what the program should do
5. Use the most obvious fix (fix the symptom!)

The Devil's Guide To Debugging

Stolen from Steve McConnell's Code Complete

1. Scatter output statements everywhere
2. Debug the program into existence
3. Never back up earlier versions
4. Don't bother understanding what the program should do
5. Use the most obvious fix (fix the symptom!)

Better Debugging

- Use the scientific method.

Better Debugging

- Use the scientific method.
- Take your time - debugging can take more time than coding!

Better Debugging

- Use the scientific method.
- Take your time - debugging can take more time than coding!
- Set up test cases that you can work out by hand - or at least estimate.

pdb

- pdb is a Python library that lets you step through your program and access variables.

```
import pdb
pdb.set_trace() #start debugging
```

pdb

- pdb is a Python library that lets you step through your program and access variables.

```
import pdb
pdb.set_trace() #start debugging
```

- Some commands:
  1. h(elp) - print list of commands (USE THIS!)
  2. b(reak) [linenum]- set breakpoint here or at linenum
  3. cl(ear) [linenum] - delete breakpoint here or at linenum
  4. s(tep) - step into function called in this line
  5. c(ontinue) - execute until breakpoint
  6. p [exp] - print expression e.g. a variable

Demo

- Classic problem: Strip HTML tags from a string.

```
<b>foo</b> ---> foo
```

- Solution: Have a boolean TAG variable that checks if you are inside a tag or not.

Demo

- Classic problem: Strip HTML tags from a string.

```
<b>foo</b> ---> foo
```

- Solution: Have a boolean TAG variable that checks if you are inside a tag or not.

Demo Time!