7.Debugging_techniques

December 15, 2018

1 Debugging techniques

What is a debugging?

Bugs can be really easy

```
In [21]: prit('So fucking easy')
```

```
---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-21-fe4019629a7d> in <module>()
----> 1 prit('So fucking easy')


NameError: name 'prit' is not defined
```

And really tricky (I won't give you examples now)

Let's classify them * Syntax errors - classic shit: * mistyped function name * missed `:` and so on

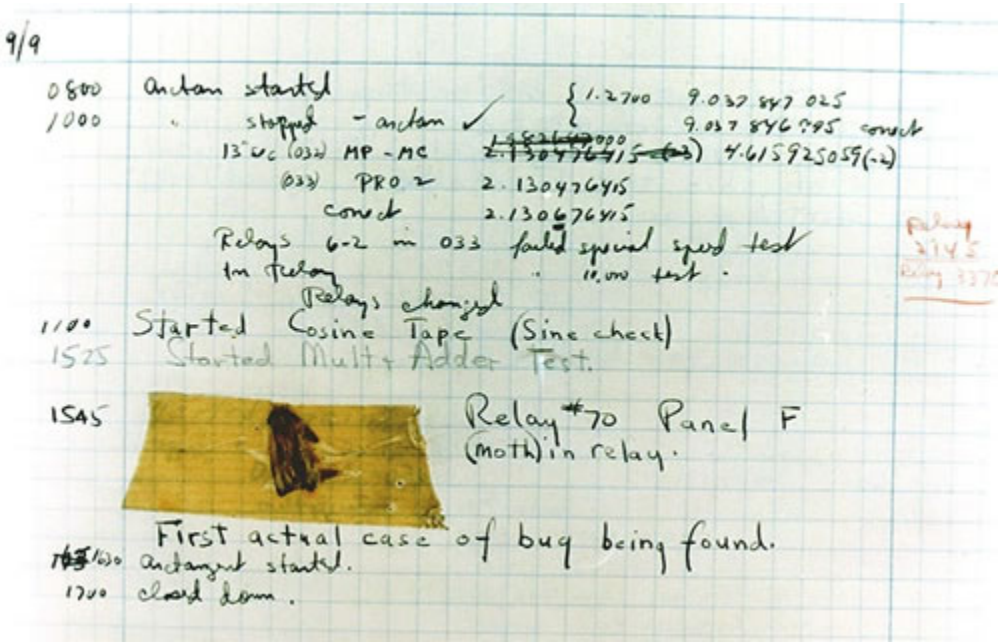
Programs with these errors failed once you try to launch it, and it is good - you get inform about this mess.

With expirience you will write better, and you will be able to just read a message to understand what's wrong and fix it, so they are easy

- "Static" runtime errors - slightly harder:
  - use `index` on empty list
  - subtract number and string
  - divide by 0 and so on

Programs with these errors failed once interpreter try to run line of code with error.

- Logic errors - fucking motherfuckers:
  - - instead of +
  - inverse predicate and so on


9/9
0800 Arctan started
1000 " stopped - arctan ✓
1.2700 9.037 847 025
9.037 846 795 correct
13" vc (032) MP - MC
4.615925059(-2)
(033) PRO 2 2.130476415
correct 2.130676415
Relays 6-2 in 033 failed special speed test
in relay
" 10.00 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.
1700 closed down.

These ones will not fail and it is awful - you need to go through your code to find it, moreover you could even don't no about their presence!

In addition to last type some bugs are random - they occur only at some conditions: * random number * user input

1.1 Main question of this lecture - How to deal with bugs

Well you have several options. Bad one is passively staring into code.

Good ones: 1. Debugging prints 1. Debugger 1. Rubber Duck

1.1.1 prints

Just add `print` statements at points of your program where something is changed

```
In [8]: # Number of cakes
        cakes = 0

        # Baker cook 3 cooks each day
        for i in range(10):
            cakes += 3

        # Your grandmother suddenly cooked 15 cakes
        cakes -= 15

        cakes
```

```
Out[8]: 15
```

```
In [13]: # Number of cakes
         cakes = 0

         # Baker cook 3 cooks each day
```

```
for i in range(10):
    cakes += 3
    print('Baking iteration number', i, '-', cakes, 'cakes')

# Your grandmother suddenly cooked 15 cakes
cakes -= 15
print('After grandmother gift -', cakes)

cakes
```

```
Baking iteration number 0 - 3 cakes
Baking iteration number 1 - 6 cakes
Baking iteration number 2 - 9 cakes
Baking iteration number 3 - 12 cakes
Baking iteration number 4 - 15 cakes
Baking iteration number 5 - 18 cakes
Baking iteration number 6 - 21 cakes
Baking iteration number 7 - 24 cakes
Baking iteration number 8 - 27 cakes
Baking iteration number 9 - 30 cakes
After grandmother gift - 15
```

```
Out[13]: 15
```

1.1.2 Debuggers

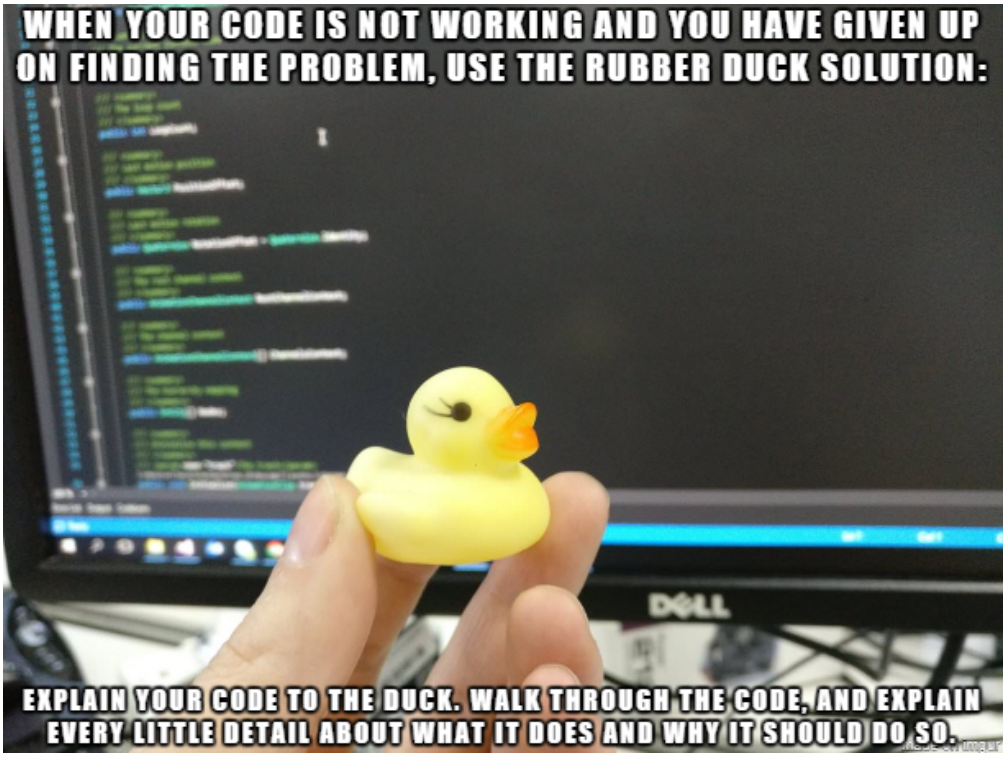
Important part of debugging - visualize value of all variables. Similar to pythontutor but works with more complex programs

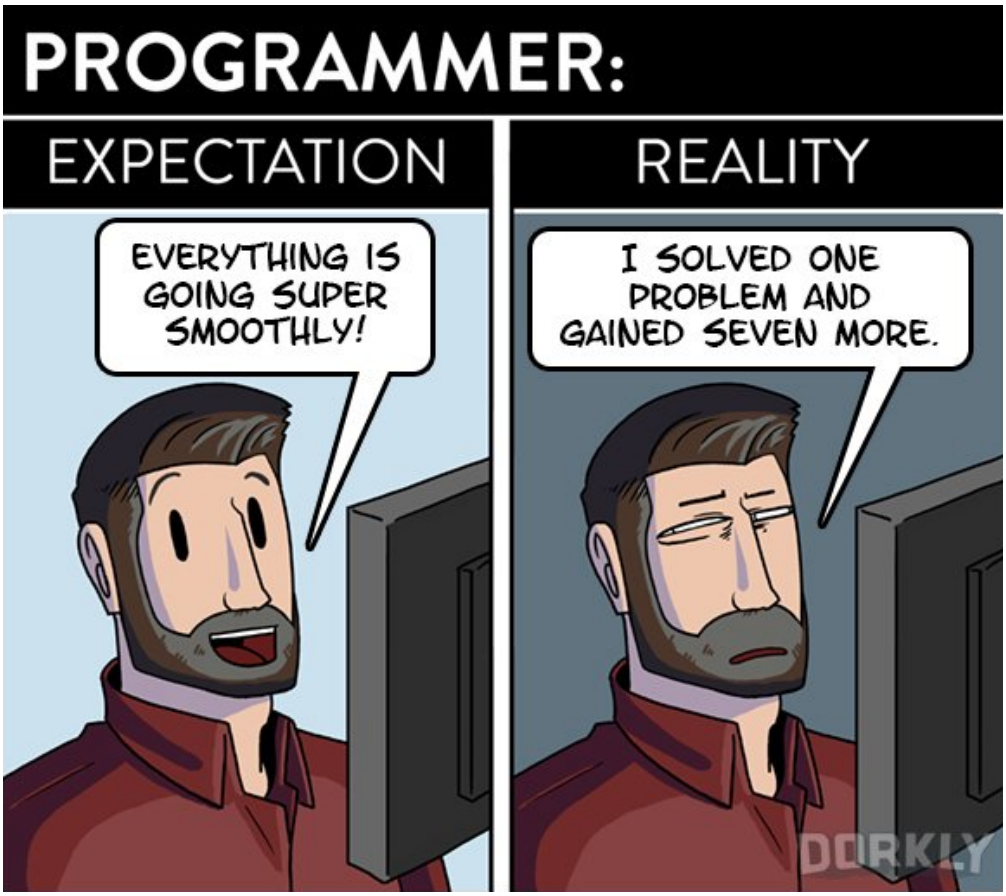
1.1.3 Duck

And the best method

It is quite a usual situation to create bugs after fixing ones due to system complexity - imagine changing of hormone concentration in organism

What to do with it? We will talk about it later)


WHEN YOUR CODE IS NOT WORKING AND YOU HAVE GIVEN UP ON FINDING THE PROBLEM, USE THE RUBBER DUCK SOLUTION:
EXPLAIN YOUR CODE TO THE DUCK. WALK THROUGH THE CODE, AND EXPLAIN EVERY LITTLE DETAIL ABOUT WHAT IT DOES AND WHY IT SHOULD DO SO.


PROGRAMMER:
EXPECTATION
REALITY
EVERYTHING IS GOING SUPER SMOOTHLY!
I SOLVED ONE PROBLEM AND GAINED SEVEN MORE.
DORKLY