debugging

July 1, 2017

1 Debugging Python Programs

Material download: github.com/krother/talks
Dr. Kristian Rother
www.academis.eu

2 Goal: are monkeys and bananas genetically similar?

Created by Oksana Latysheva from the Noun Project
Created by Vladimir Belochkin from the Noun Project

3 Goal: are monkeys and bananas genetically similar?

3.1 Input: Protein sequences (strings)

```
>tr|A0A075B6H5|A0A075B6H5_HUMAN T cell receptor..
METVVTTLPREGGVGPSRKMLLLLLLLGPGSGLSAV
...
```

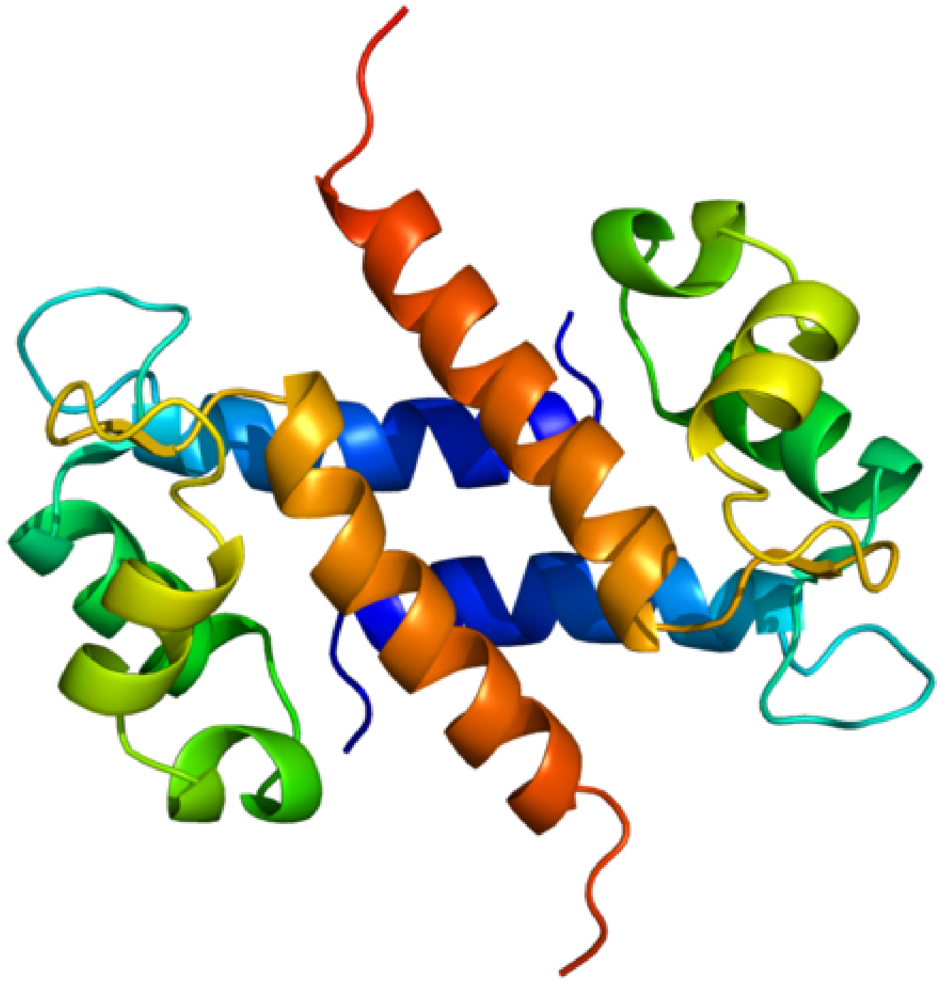
3.2 Output

Average character count in **chimp**, **banana** and **human** (as reference).

4 What is a protein?

Proteins are tiny molecular machines that do all kinds of things in living cells. For example, antibodies, digestive enzymes and spider silk are all made of protein.

Proteins are **chains made of 20 chemical building blocks**, which is why we can easily represent and analyze them as strings.



Protein S100A8 PDB 1mr8 by Emw - Own work, CC BY-SA 3.0

5 Task: execute `code_buggy/parse_uniprot.py`

5.1 Fix all bugs

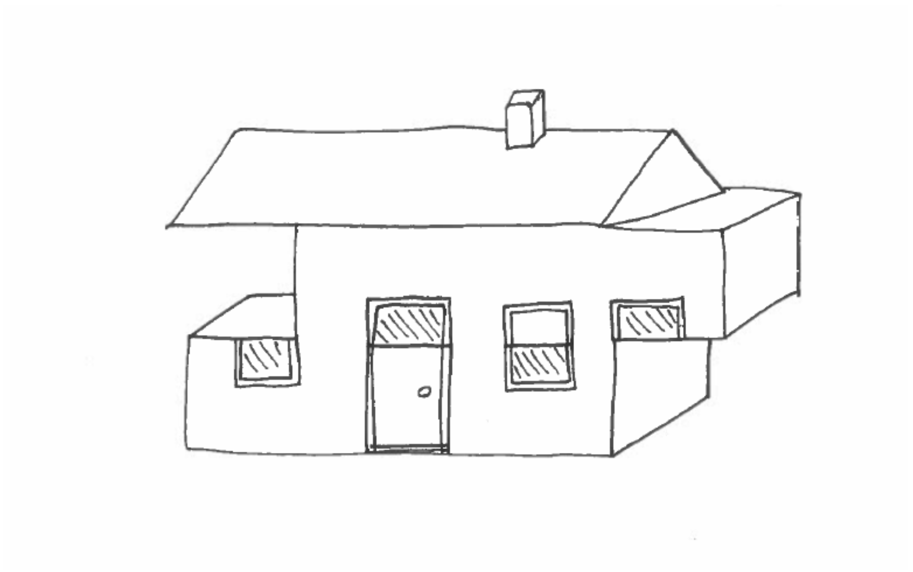
6 Technique #1: Read the Error Message

```
parse_uniprot.py
  File "parse_uniprot.py", line 65
```

```
    for aa in AMINO_ACIDS
                        ^
SyntaxError: invalid syntax
```

Rule of thumb: read Python error messages from bottom to top.

7 SyntaxError: Something obvious is wrong



8 Exceptions at Runtime

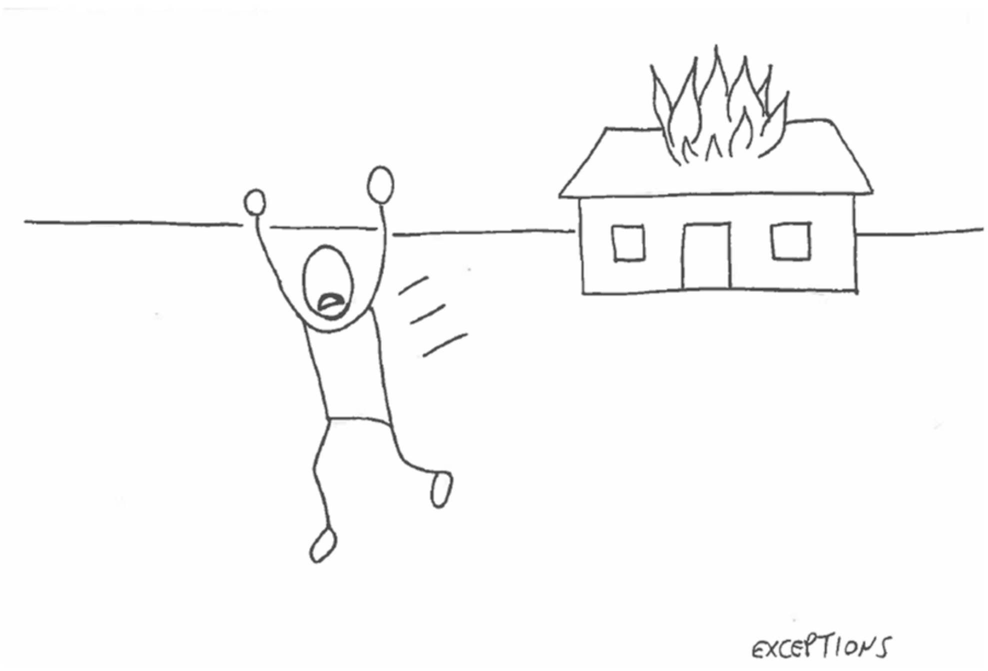
9 Python is not the easiest language to debug

```
In [2]: data = [1, 2, 3, 4
```

```
      File "<ipython-input-2-516badf311e1>", line 1
    data = [1, 2, 3, 4
                      ^
SyntaxError: unexpected EOF while parsing
```

```
In [3]: data = 1, 2, 3, 4]
```

```
      File "<ipython-input-3-f2ba22b1c99a>", line 1
    data = 1, 2, 3, 4]
```



```
                    ^
    SyntaxError: invalid syntax
```

10 Nasty fact: Errors propagate

11 Technique #3: Minimize the input

our input file is TOO BIG!

Create a smaller file with just 3 entries, e.g.:

```
> tr | ABC12345 | python fake protein (Os=Homo sapiens) tail
PYTHQNMMXVII
```

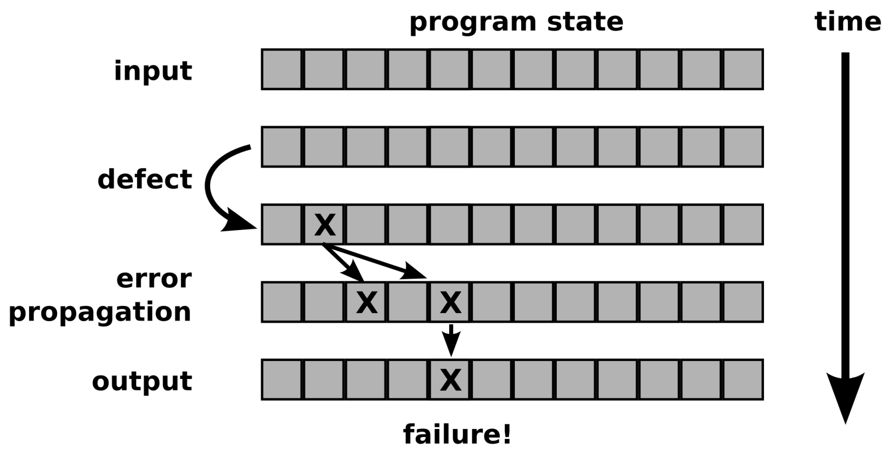
12 Technique #4: Scientific Method

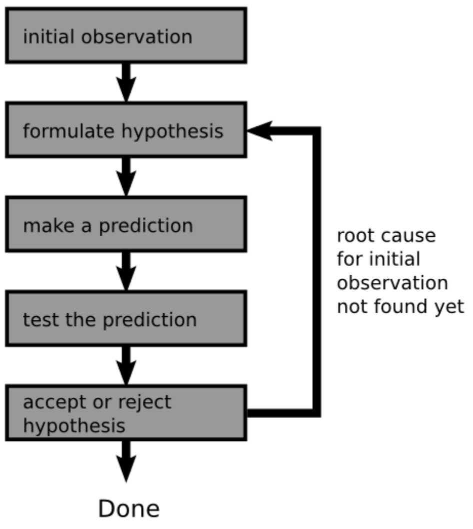
13 Technique #5: Add print Statements

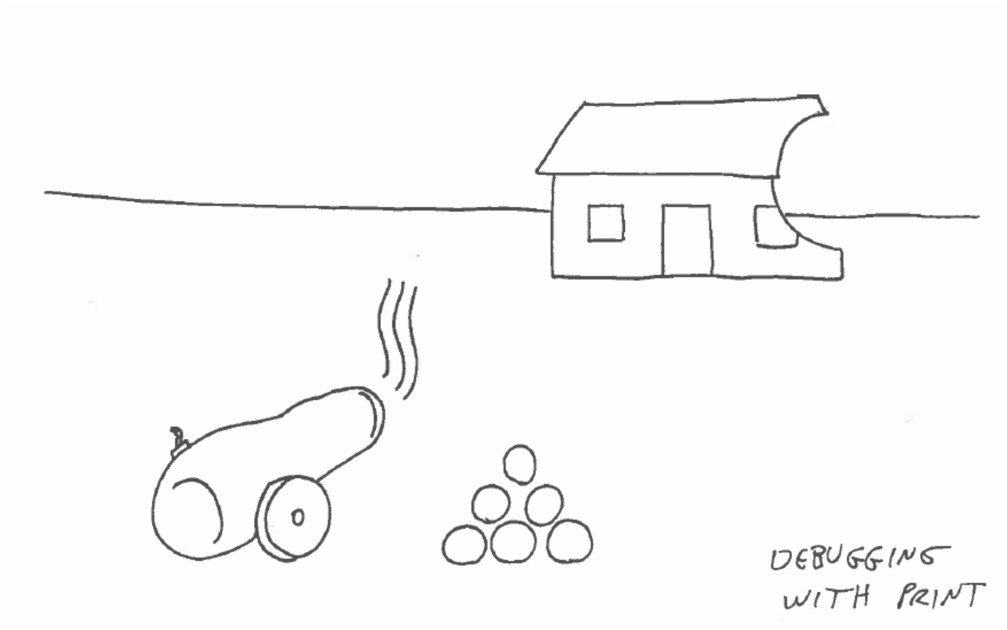
`print()` is a bit like shooting holes into a wall to see what is inside

14 Introspection functions

more elegant alternatives to `print()`:


program state
time
input
defect
error propagation
output
X
X
X
X
failure!


initial observation
formulate hypothesis
make a prediction
test the prediction
accept or reject hypothesis
root cause for initial observation not found yet
Done



print statements

function	purpose
`dir(x)`	examine local namespace
`locals(x)`	examine local namespace
`globals(x)`	examine global namespace
`help(x)`	access help interactively
`type(x)`	examine object type
`isinstance(x, cl)`	examine object type
`issubclass(cl1, cl2)`	examine class hierarchy

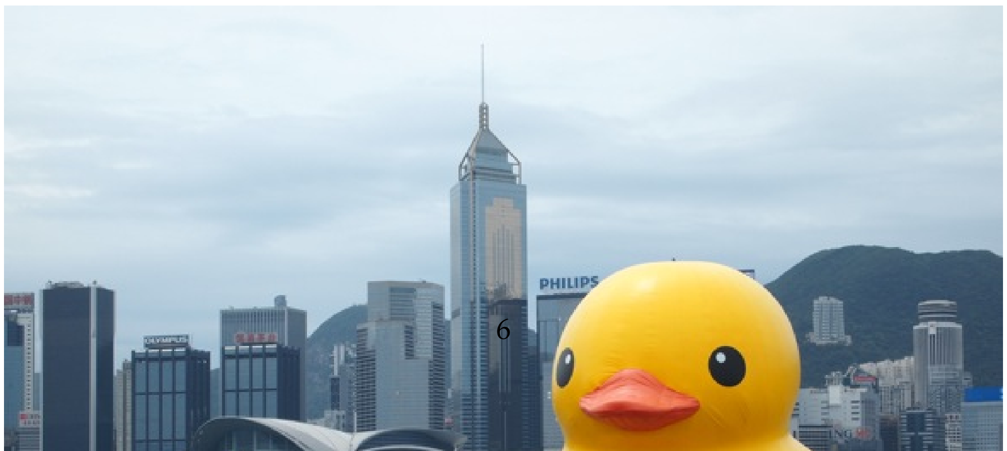
15 Technique #6: Assertions

We creating a consistency check for **failing early**. Add the following assertion:

```
assert sum(aa_counts) == len(seq)
```

in `parse_uniprot.py` at the end of the `parse()` function.

16 Technique #7: Explain the problem to someone



17 Technique #8: Interactive Debugger

An interactive debugger allows us to watch our program at work in slow motion

```
In [ ]: import pdb
        pdb.set_trace()
```

```
--Call--
> /home/krother/anaconda3/lib/python3.6/site-packages/IPython/core/displayhook.py(236)__call__
-> def __call__(self, result=None):
```

18 Stepwise Execution in ipdb

command	description
`l`, `ll`	list lines
`n`	execute next line
`s`	step into function
`c`	continue execution
`q`	abort
`?`	see other commands

19 Breakpoints in ipdb

command	description
`b`	list breakpoints
`b <file:line>`	add breakpoint
`b <function>`	add breakpoint
`b <file:line>, <condition>`	add breakpoint with condition
`cl <number>`	remove breakpoint

20 Technique #9: Code Review

Conduct a code review of `pipeline.py` with your neighbour.

- Which part of the code is clear to you? Which is not?
- Do you find any bugs?
- What would you improve in the code?

21 Code Review

The control mechanism of the lock of a vault for nuclear waste has been designed for safe operation. It makes sure that it is only possible to access the vault, if the radiation shields are in place or the radiation level in the vault is below a threshold (DANGER_LEVEL). That means:

- If the remote-controlled radiation shields are in place, the door may be opened by an authorized operator.
- If the radiation level in the room is below the threshold, the door may be opened by an authorized operator.
- An authorized operator may open the door by entering a code.

The code below controls the door lock. Note that the safe state is that no entry is possible. Develop an argument for safety that shows that the code is potentially unsafe.

(adopted from *I.Sommerville, Software Engineering, 9th edition*)

Trivia: Code reviews are seen as superior to automated testing when engineering ***safety-critical software****.*

22 Code Review

Review the following (fictional!) code for a nuclear vault door:

```
entry_code = lock.get_entry_code()
if entry_code == lock.authorised_code:
    shield_status = shield.get_status()
    radiation_level = rad_sensor.get()
    if radiation_level < DANGER_LEVEL:
        state = SAFE
    else:
        state = UNSAFE
    if shield_status == shield.in_place():
        state = SAFE
    if state == SAFE:
        door.locked = False
        door.unlock()
    else:
        door.lock()
        door.locked = True
```

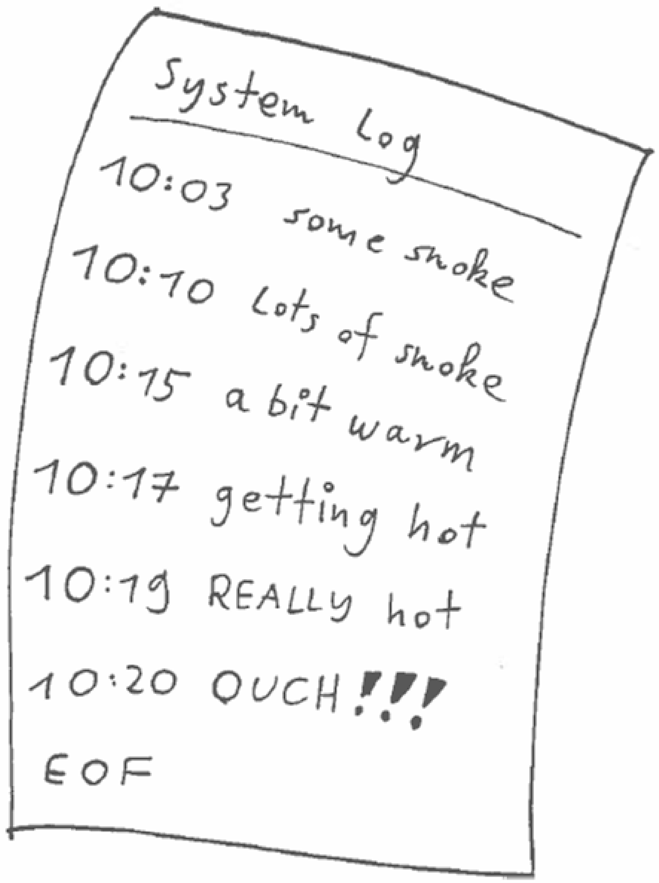
(code adopted from *I.Sommerville, Software Engineering, 9th edition*)

23 Technique #10: Log information

24 Log verbosity levels

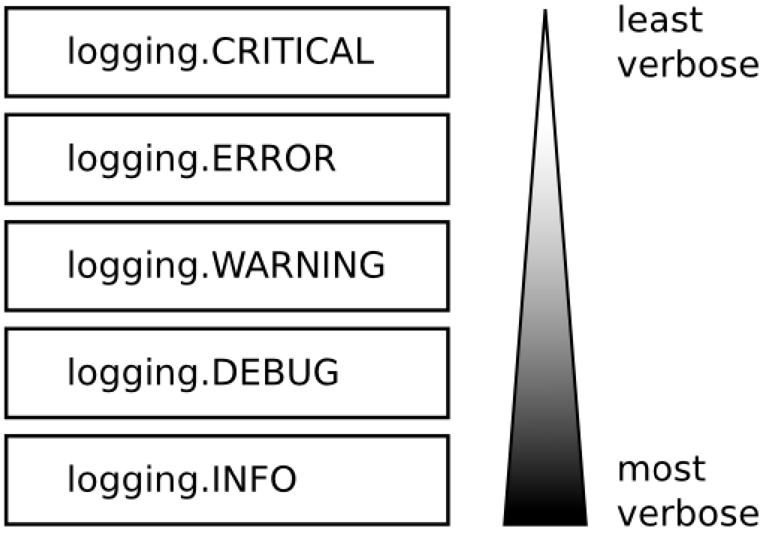
25 Summary: What we know about debugging

- error messages in Python are not always helpful
- syntax errors are when Python does not do anything
- some errors cause a program to stop with an Exception
- read error messages from bottom to top
- semantic errors: the program does not do the right thing
- errors are distinct from the underlying defects
- defects propagate through the program



```
System Log
10:03   some smoke
10:10 Lots of smoke
10:15  a bit warm
10:17 getting hot
10:19 REALLY hot
10:20 OUCH!!!
EOF
```

log



26 10 Debugging Techniques

1. **read the error message** first
2. use **minimal input** for diagnosing bugs
3. **reproduce** the bug
4. use the **scientific method** instead of guessing blindly
5. add **print statements** to get diagnostic information
6. **assertions** check for consistency
7. **explaining the problem** to someone often helps
8. use an **interactive debugger**
9. **code reviews** are a superior method for creating reliable software
10. **logging** creates more diagnostic information

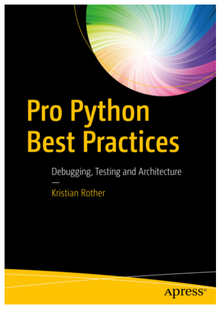
27 More on Debugging

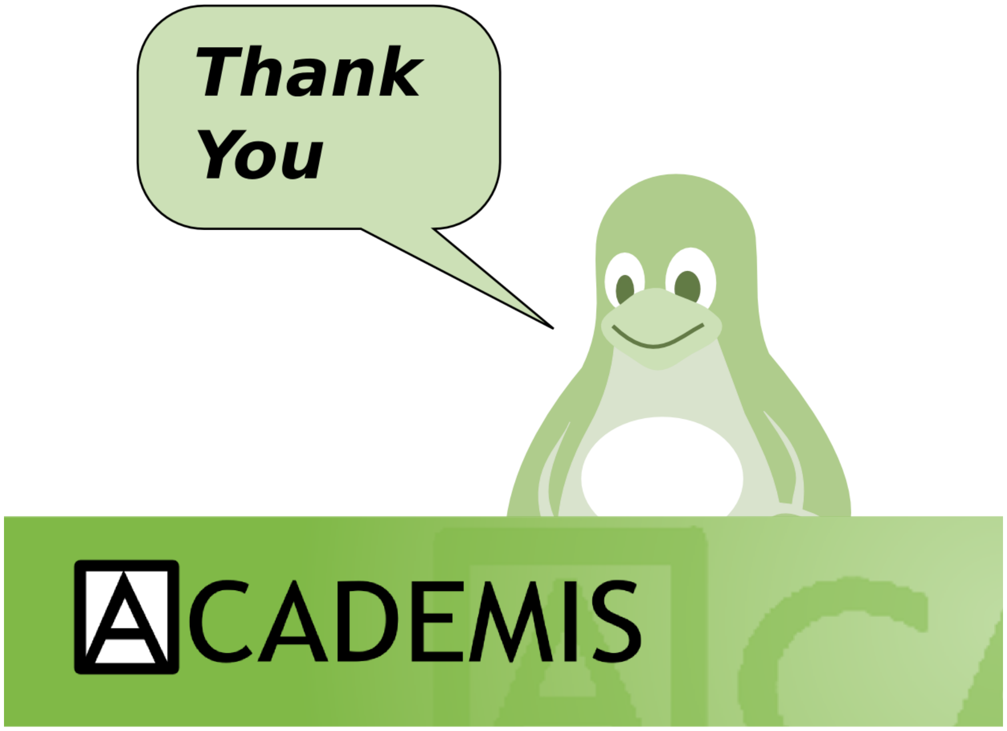
Pro Python Best Practices (Apress, 2017)

28 More on Python: www.academis.eu

Contact: krother@academis.eu

```
In [ ]:
```


Pro Python Best Practices
Debugging, Testing and Architecture
Kristian Rother
Apress


Thank You
ACADEMIS

Thank you