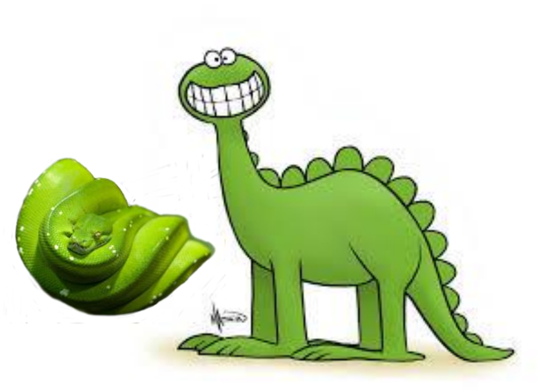
A Dinosaur and a Python Walk into a Bar. . .

How a dinosaur debugs in Python



Richard Landau
Retired Software Engineer
 & Boston Python junkie

Who're You Calling a Dinosaur?!?!

- Me
  - Retired SW engr
- Used fifty (!) programming languages
  - Python is #1
    - Concise, expressive, astonishing libraries, iPython/Jupyter, ...
- Develop on Windows/Cygwin, run production on Linux

Topics

- Main topic: offline debugging
- Lessons learned over the years
  - What I use and why

Offline Debugging

- Write a record of what a program does
  - **Not** as a permanent record
  - **Not** as an activity log for auditing

Interactive Debugging Might Not be Simple

- Might not be possible or practical
  - Detached process with no UI access
    - E.g., part of a web service
  - Lengthy or complex setup
  - Long run time

Need an Alternative

- Need an adequate method of watching what a program does/did
  - Tracing the code flow (squint, not detail) and
  - Tracing the data progress

- For me, faster debug for most programs

New Program Doesn’t Do What You Want It To Do?

- Sprinkle a few print() statements on it
- Do better

A Modest Example Here

- NewTrace area on github.com/rblandau
- Like print() on steroids
- Plus other features from experience

Ned is Holding His Nose

- He saw my crude 2007-8 Python code
  - "Python for Beginners" version
    - Not even decorators
- Back then I was writing "C without braces"

Lessons from Experience

- Important things to include, based on experience
  - 1978-2019
    - This trace facility written in PDP-11 assembler, then C, C++, VB, Perl, Java, and finally Python
    - Originally written for a multiprocess comm system

First Step Beyond print()

- Sample decorator: print function entry and exit, in every textbook

- **<code: decorator example>**

- Very useful, but can be improved

Lesson: Output Must Have

- Time
- Source location
- Activity
- Data content
- Consistent and compact format
- stdout/stderr or file output

Lesson: Nice to Have

- Adjustable level of detail (priority)
- Optional HTML output
- Low performance impact
- Thread-safe, multiprocessing-safe

Lesson: EZ2 Manage

- **Manage without editing** code
  - On/off
  - Change output location
  - Adjust filter by detail level, source location

Currently Try to Meet

- Standard, compact format
- Function enter/exit or anywhere
- Output location and format
- Filter by importance, source
- Low performance impact

and

- **Manageable without editing code!**

Timestamp

- Date and time, to second or millisecond
- (All examples from real code, real output)
- **<code: timestamp>**

Easy Control of Debug On/Off

- Control with **environment variables**
- Environment vars easy to sense
- (Restart process when params change)

- **<code: environment variable control>**

Lesson: Keep the Debugging Code

- No separate debug/nondebug versions
- When customer calls with a problem...
  - Did you leave tracing permanently installed?
    - Just tell the customer to add a couple environment variables and restart
    - Send me the log file
      - Yes, we actually did that

Output Where?

- Send to stdout if possible
  - 'less' utility
  - Need file output, too, if stdout not available
  - Optional HTML tags
  - Maybe more than one output at a time, e.g, stdout and file

- **<code: looking at output>**

Low Performance Impact

- Without editing the source code
  - Environment var eliminates almost all code
    - Null decorators
    - Almost-null direct ntrace calls
  - Recompile if you change production on/off

- **<code: if production mode>**

The Package I Use

- One file, one class
  - Functions to write traces
  - Decorators that use the functions
- Singleton instance of the class

Why Singleton Instance?

- Efficiency
  - Evaluate conditional code at compile time
  - Read run envir var params only once
- Less for programmer to write

- **<code: imports>**
- **<code: singleton>**

Trace Functions

- ntrace(priority, outputline)
- ntracef(priority, facilityname, outputline)

- Instance of class to access these functions

- **<code: NTRC examples>**

Decorators

- @ntrace
- @ntracef(facilityname, priority)

- **<code: decorator examples>**

Filtering on Priority, Facility

- Filtering makes listings short and to the point, when needed
- Envir vars
  - TRACE_LEVEL (int)
  - TRACE_FACIL (string)

Detail Levels I Use

- TRACE_LEVEL environment variable
  - 1 for enter/exit function
  - 3 for most data details
  - 5 for excruciatingly detailed details

-
  ```
  export TRACE_LEVEL=3
  python ...
  unset TRACE_LEVEL
  ```

Filter by Facility Names

- TRACE_FACIL environment variable
  - Default = ALL
  - ALL, NONE, +, -
    - ALL-FOO-BAR
    - NONE+FOO+BAR
    - Unnamed trace lines always print

-
```
export TRACE_FACIL=NONE+FOO
python ...
unset TRACE_FACIL
```

Lesson: Identify Data, Not Just Location

- Class instances look alike
- Identifying instances by address or id()?
  - – Not useful to humans
  - – Solution: create names for instances

Map IDs to Instances

- Store a unique ID in each instance
  - Pass ID strings in argument lists
    - Not just instances (addresses)
  - Use dictionaries to map from (class and) ID-string to instance
    - Dictionary lookup is cheap

- **<code: instance identifiers>**

Contact

- www.github.com/rblandau/NewTrace

- email: landau@ricksoft.com

Other Tidbits

Use for Regression Testing

- Tests can be done with log files suitably sanitized
  - Timestamps and other non-deterministic items
  - (Ordering tricky if multi-threaded or multiprocessing)
- Possible with trace files in some cases

Python trace Module?

- The standard Python trace module doesn't fit my needs
  - Tracks process, sequence of statements

Multiprocessing

- Multiprocessor-safe, or at least thread-safe, would be good
- Not yet

Use Meaningful Names

- The type of the data should be obvious
  - It prevents a lot of runtime errors
- I use "Hungarian naming" because (dinosaur!) I grew up with it
  - CamelCase with ugly prefixes for datatypes
  - CamelCase is just a personal preference BUT
  - Datatype at the beginning of the name of data (or a function) is useful

Sort of Apology

- Not PEP-8, sorry
  - But you can embed type in a pep8 name, too
    - foo_list or list_foo
- If I were building packages for distribution, I would reform my evil ways

"Any programmer who fails to comply with the standard naming, formatting or commenting conventions should be shot. If it so happens that it is inconvenient to shoot him, then he is to be politely requested to recode his program in adherence to the above standard."

-- Mike Spier, Digital Equipment Corporation, 1971

Meaningful Data Contents, Too

- If you have lots of class instances, addresses are not useful to humans
  - Assign readable IDs in instances
- My last project had only ~20 major classes but 10-50,000 instances of some classes
- (The debug package here displays IDs, instance names)