Debugging & Profiling Code, in Python and R

Abhishek Biswas

Research Software Engineer, Princeton University

ab50@princeton.edu

Wintersession / PICSciE Training Workshops

Attendance : http://cglink.me/2gi/c1340958101282849

CHECK-IN

DEBUGGING & PROFILING CODE, IN PYTHON AND R

FRIDAY, JANUARY 14 AT 1:00PM

Objectives

- Overview of code structuring and debugging
- Overview of Integrated Development Environments (IDE)
  - Python – PyCharm
  - R – RStudio
- Debugging typical software errors
  - Program crashes, Incorrect output
- Profiling
  - Inefficient resource utilization (Time, Memory, CPU)

Design is Important

- Good code design will make finding bugs easier
- Easier to track down bugs in modules
  - Easier to write targeted testing code
- Easier to replace and fix the bug
  - Less chances of introducing more bugs

Print Statements and Comments

- Printing and logging messages are the most basic and useful tool
  - Can’t figure out how to setup a debugger
  - Parallel execution
  - Help isolate the portion of the code
- Commenting out code
  - Comment out portions of code and see if it runs
  - Again, helps isolate the problem

PDB – commandline debugging

- Program crashes produce a stack trace with line numbers

```
Traceback (most recent call last):
 File "no_furniture.py", line 16, in <module>
   for item in items.sort():
TypeError: 'NoneType' object is not iterable
```

- Python provides a command like debugger too
  - pdbpp is a more user-friendly version

- Let’s do the first demo!

Breakpoints, Watches and Evaluation

- Breakpoints allow you to halt the flow of execution
  - Stop a few lines before the error
  - The error in most cases happens before the line that triggers it
- Watches allow you to look at the values of the variables
- Evaluate allows you to insert and run new code to check things

IDEs Help Design and Navigate

- Help you manage your files
  - Separate out modules
  - Clearly save src, test and metadata files
- They help you navigate through your code
  - Learn the shortcuts
- Help you setup debugging environment
- Help you setup build and release processes

Exercise 1: PyCharm Setup

- Download Python Project:
  - Link: https://tinyurl.com/5fc4byb6
  - Download and extract the project directories
- Start Anaconda Navigator and launch PyCharm
  - Go to File > New Project
  - Navigate to the “python_debug” project directory
  - **Create project using existing files**
- Understand project structure
  - Set source root as the “src” directory (right click option)

Exercise 2 : Fix The Tests!

- Fix the failing test cases…
- Useful links
  - https://byjus.com/maths/area-of-shapes
  - https://byjus.com/volume-formulas
  - https://byjus.com/surface-area-formulas

Conditional Breakpoints

- Sometimes breakpoints need to be conditional
  - Useful when breaking at a certain iteration
  - Breaking on a certain call of the function
- Run the “triangle_area_many.py” example
  - Figure out which of the 10000 triangles is failing!

Exercise 2: Profiling Python Code

- Create a new python project using to “**python_profile**” directory

- python -m cProfile -s **tottime** search.py
  - simple_search
  - sort_search
  - best_search

- Why is simple_search faster?
  - Profile the code
  - What is taking so long?

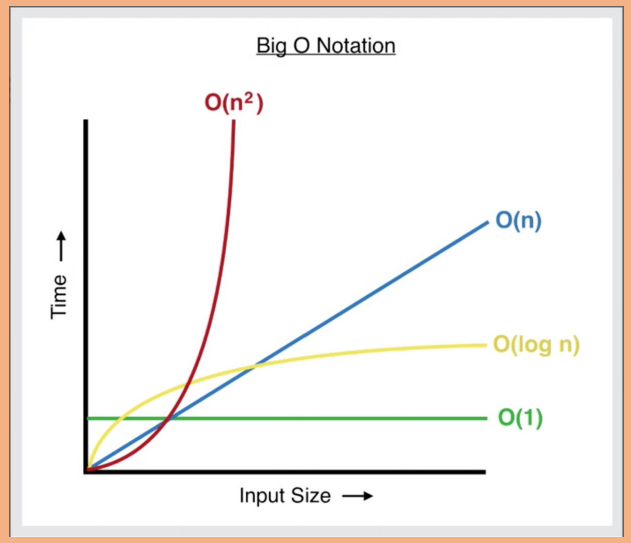
Valid Arg	Meaning
`'calls'`	call count
`'cumulative'`	cumulative time
`'cumtime'`	cumulative time
`'file'`	file name
`'filename'`	file name
`'module'`	file name
`'ncalls'`	call count
`'pcalls'`	primitive call count
`'line'`	line number
`'name'`	function name
`'nfl'`	name/file/line
`'stdname'`	standard name
`'time'`	internal time
`'tottime'`	internal time

Did anyone run the tests for search?

Check correctness before profiling!

Generating Growth Curves

- Profiling is used to generate growth curves



- Let's look at 3 sorting algorithms
  - python sort.py

Python Memory Profiling

- Memory profiling show the instructions that significantly increase the memory usage
- memory_profiler module allows users to decorate functions
- Shows total memory usage and lines that add to memory footprint
  - python mem_profile.py

Exercise 3 : Rstudio Debugging

- Download the file:
  - Navigate to the R_debug folder
- Click on the files to open Rstudio
  - Click on **Source** button to run the scripts
- Try to fix the infinite loop!
  - random_debug.R

Exercise 4 : Rstudio Profiling

- Download the file:
  - Navigate to the R_profile folder
- Click on the files to open Rstudio
  - Click on **Source** button to run the scripts
- Let’s benchmark some sorting and dataframe operations

Profvis Interactive Profiling

- Produces an interactive graph
- Statistical profiling using Rprof
  - 10ms intervals
- The flame graph shows the lines of code where time was spent

Have a great weekend!