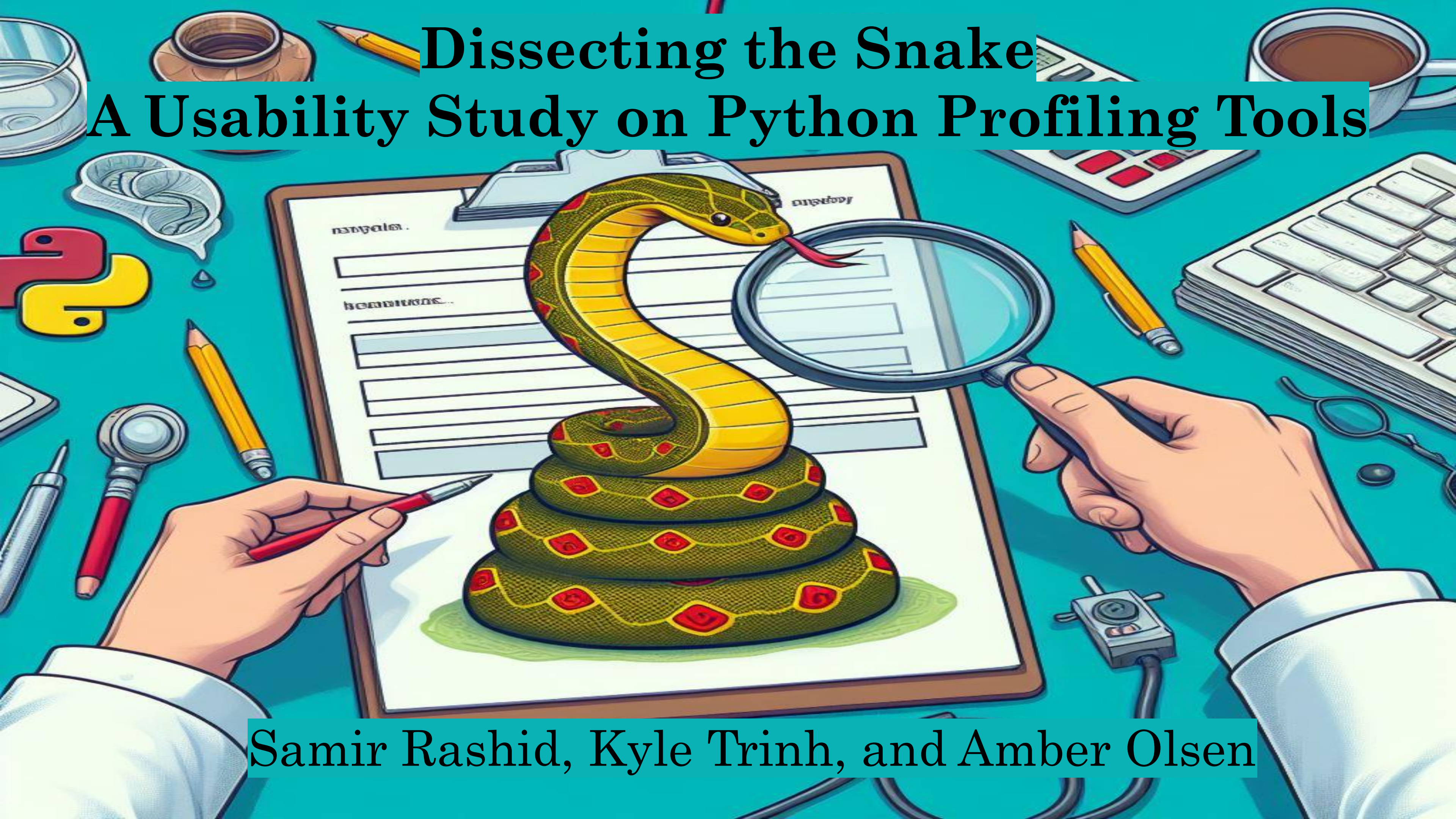
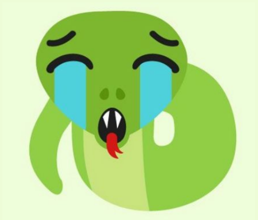
Dissecting the Snake
A Usability Study on Python Profiling Tools



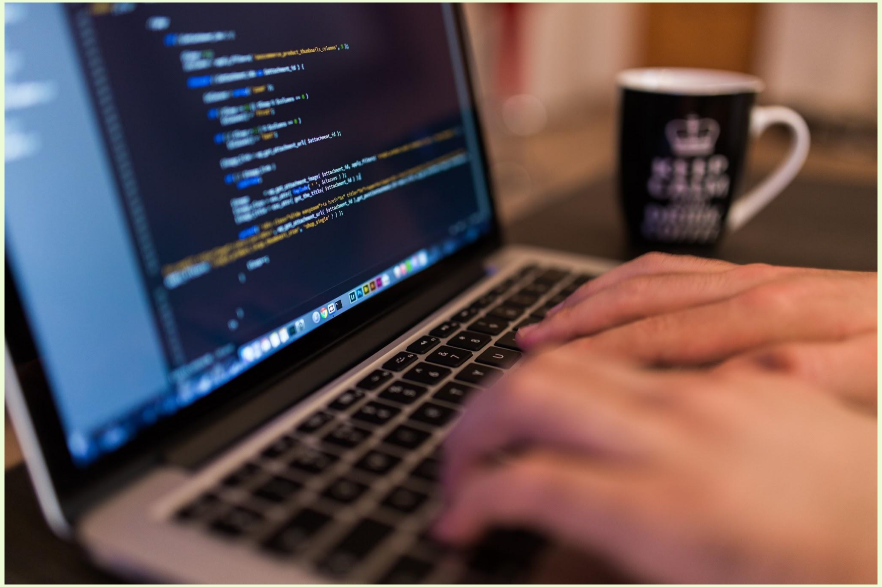
Samir Rashid, Kyle Trinh, and Amber Olsen

Introduction



Optimizing code during runtime is an essential part of programming. Profiling tools can be unwieldy and confusing to use.

In our study, we compare cProfile, the standard command-line Python profiler, to pytrail, our custom, line-by-line in-IDE profiler, pytrail. We sought to discover if having line-by-line information about the efficiency of code would help programmers

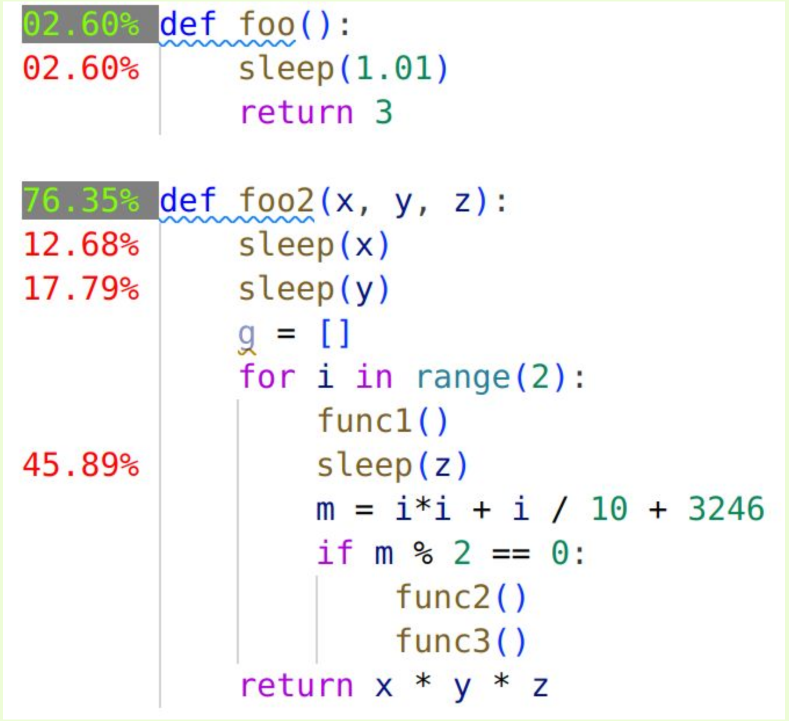


Research question

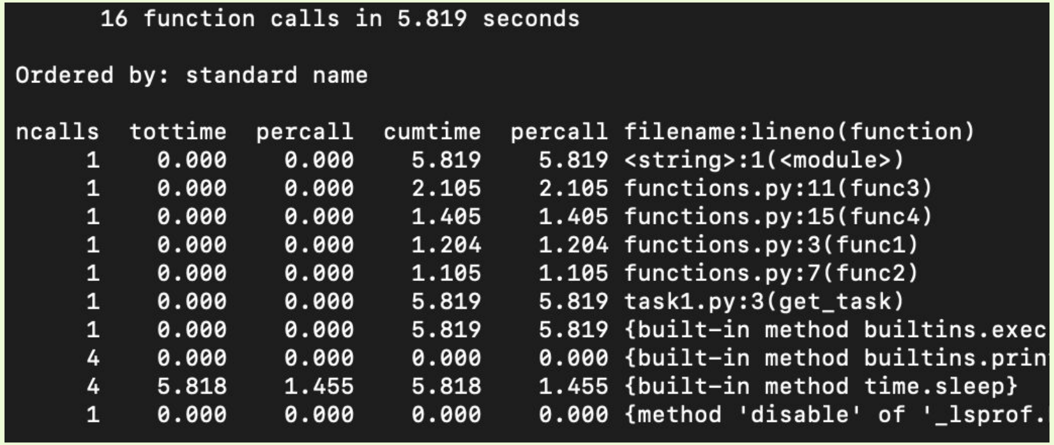
Does integrating a Python profiling tool into an IDE help Python programmers of varied experience to detect unoptimized code more efficiently?

In essence, this question can be subdivided into two parts: does in-IDE profiling data help users write more efficient code generally, and does pytrail specifically help to this end?

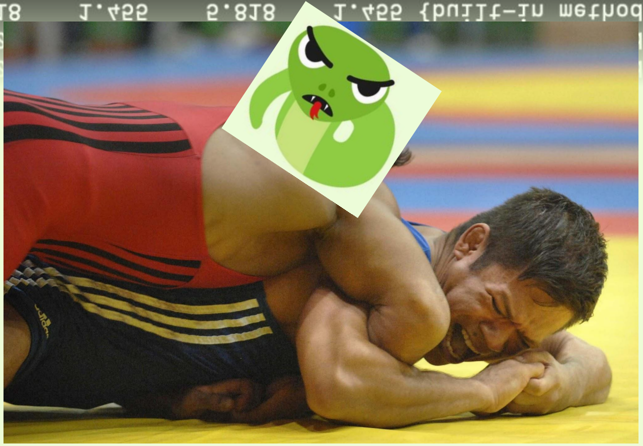
pytrail vs. cProfile



```
02.60% def foo():
02.60%     sleep(1.01)
           return 3

76.35% def foo2(x, y, z):
12.68%     sleep(x)
17.79%     sleep(y)
           g = []
           for i in range(2):
               func1()
45.89%         sleep(z)
               m = i*i + i / 10 + 3246
               if m % 2 == 0:
                   func2()
                   func3()
           return x * y * z
```



```
         16 function calls in 5.819 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    5.819    5.819 <string>:1(<module>)
        1    0.000    0.000    2.105    2.105 functions.py:11(func3)
        1    0.000    0.000    1.405    1.405 functions.py:15(func4)
        1    0.000    0.000    1.204    1.204 functions.py:3(func1)
        1    0.000    0.000    1.105    1.105 functions.py:7(func2)
        1    0.000    0.000    5.819    5.819 task1.py:3(get_task)
        1    0.000    0.000    5.819    5.819 {built-in method builtins.exec
        4    0.000    0.000    0.000    0.000 {built-in method builtins.prin
        4    5.818    1.455    5.818    1.455 {built-in method time.sleep}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.
```



Recruitment & Methodology of Tool Effectiveness Study

Recruitment Method

- Utilized Piazza, an educational forum, with an incentive of a $10 drawing entry for participants.
- Two Python scripts, task1.py and task2.py, were used for analysis.

Procedures

- Participants were assigned either cProfile or pytrail to analyze task1's runtime.
- For task2, they used the tool not previously used.

Feedback Collection:

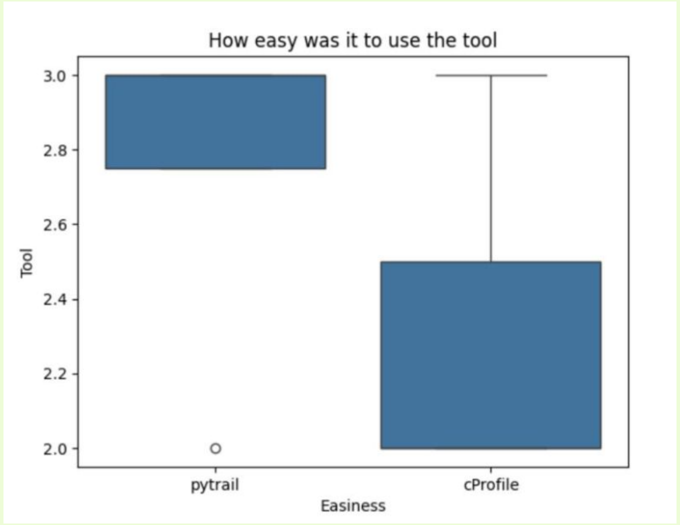
- In situ-task Google Form to gather insights on specific runtime elements (e.g., slowest code line/function).
- 10-minute limit survey for analyzing each task.

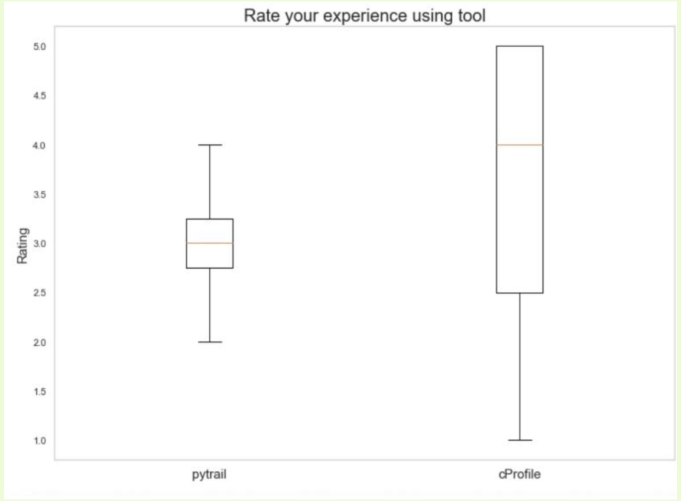
Post-Analysis Survey:

- Another Google Form to assess user opinions on each tool, focusing on ease of use, issues encountered, and potential improvements.

<u>Responses were analyzed to determine the effectiveness of cProfile and pytrail tools.</u>

Results

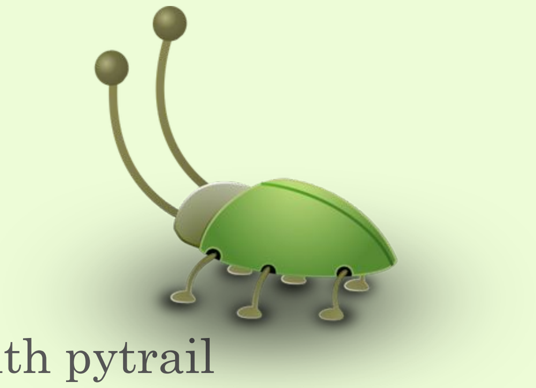




We did not find statistically significant results.

- Ease of use pytrail vs. cProfile **p=0.35**
- Rating between pytrail and cProfile **p=0.65**

Limitations



- Didn't resolve all bugs with pytrail
- Some users manually determined slowest lines of code and did not rely on either profiler
- Three out of five participants left unanswered questions in the assessment and/or after-study survey
- For one participant, we didn't give them an assessment.
- Low participation

Further Discussion

Challenges Faced:

- Portability issues
- Unresolved bugs in pytrail at the time of participant release.
- Limited time considered a factor in pytrail's shortcomings.

User Study Insights:

- Participant P1 experienced difficulties in managing code analysis and survey questions.
- Confusion among participants about runtime analysis in pytrail.
- Fundamental limitations: only measures time in functions and external library calls, not in source code calls.

Opportunities for Tool Improvement:

- cProfile found unintuitive; suggestions for language-aware enhancements.
- Need for more user-friendly tools with visual feedback, beyond command-line interfaces.
- Possibility of instant feedback tools benefiting both novice and experienced Python programmers.

Future Work

Pre-Experiment

- Add more functions to pytrail for clear indication of code annotation completion.
- Implement a pre-survey for participant screening to ensure a representative sample while managing costs via GitHub Codespaces.

Task Design

- Shift from quantity to quality of tasks, focusing on increasing complexity to better mimic real-world scenarios.

Additional Research

- Investigate the impact of integrating profiling tools into IDEs on programmers' efficiency, particularly for those unfamiliar with such tools.
- Explore user experience aspects of Python profiling tools, a relatively unexplored area in contrast to their technical capabilities.

Questions?