Python 3 Performance

Measuring Performance

Dan Tofan

Software Engineer, PhD

@dan_tofan | www.programmingwithdan.com

Python 3 Performance

Version Check

Version Check

This course was created by using:

- Python 3.11
- Visual Studio Code 1.73.1
- Pytest-benchmark 4.0
- Line-profiler 4.0
- Memory-profiler 0.60
- PyCharm Professional 2022
- Dask 2022

Version Check

This course is 100% applicable to:

- Python 3.11 and later
- Any version of Visual Studio Code from 1.73.1

Module Overview

- Understanding performance
- Strategy to improve performance
- Basic ways to measure performance
- Why profile?
- More profilers
- Visualizing profiling data

Globomantics Storyline

Happy about Python

Slow

Needs better performance

Why Performance Matters?

Happy users

Happy developers

Saving money

Environment friendly

Bottlenecks

CPU

Memory

Disk

Network

Strategy to Improve Performance

Prevent

Fix

Measure

Prevent Performance Issues

Review architecture

Performance awareness

Use efficient:

- Libraries
- Algorithms
- Data structures

Fix Performance Issues

Is Python code the bottleneck?

How?

- Try faster hardware
- Ensure test coverage

Tradeoffs:

- Code is harder to understand
- Code consumes more memory
- Significant effort

Measure Performance

"If you can't measure it, you can't improve it"

Easy to make wrong guesses

Repeat many times, with various inputs

Basic Ways to Measure Performance

Task Manager on Windows

Activity Monitor on Mac

System Monitor on Ubuntu Linux

Linux Terminal

Use the time command

Use the top command

Measure Performance with Python

Use the time() function

Use the timeit module

Use the pytest-benchmark plugin for pytest

Demo

Write basic Python program

Measure performance

- Use time shell command
- Use timeit module
- Use time() function
- Use pytest-benchmark

Why Profile?

Find bottlenecks

- Detailed measuring of execution times
- Remove guessing

Pareto principle

- Small parts of a codebase have most performance impact

Types of Profiling

Event-based or deterministic	VS	Statistical
Gather data on events		Sampling the program
Lot of data		Less data
High accuracy		Approximation
High overhead		Less overhead

Profiling Modules Included in Python

The profile module

Adds significant overhead

Easy to extend

The cProfile module

Adds overhead

General purpose

Demo

Modify the Python code

Use the profile module

Use the cProfile module

Compare results

Limitations of Integrated Profilers

- Performance
- Multithreading
- Visibility inside functions
- Memory visibility

Third-party Profilers

Line_profiler

Py-spy

Scalene

Yappi

Memory_profiler

Demo

Use line_profiler

- Install line_profiler
- Decorate function with @profile
- Run: kernprof –lv file.py

Use memory_profiler

- Install memory_profiler
- Decorate function with @profile
- Run: python –m memory_profiler file.py

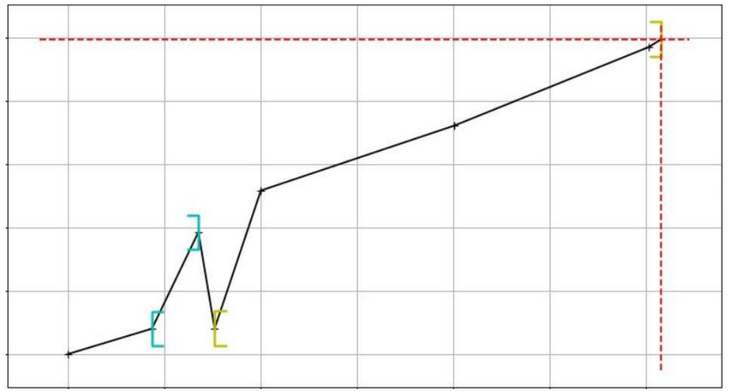
Visualizing Profiling Data

Lots of profiling data

Complex data

Need for friendly ways to visualize data

Plotting with Memory_profiler



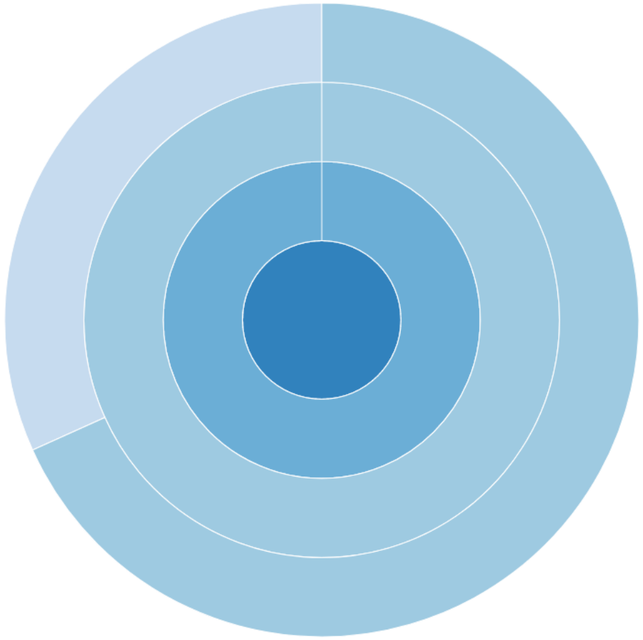
Generates plots to show memory consumption

Requires matplotlib

Usage

- Run: mprof run file.py
- Run: mprof plot --output file.jpg

Visualizing Profiling Data with Snakeviz



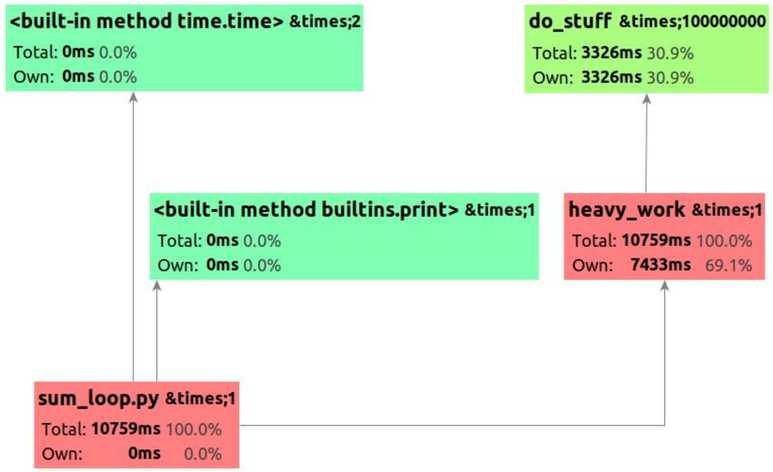
Visualize cProfile output

Browser-based

Usage

- Run: pip install snakeviz
- Run: python –m cProfile -o file.prof file.py
- Run: snakeviz file.prof

Visualizing Profiling Data with PyCharm Professional



Not available in PyCharm Community edition

View statistics

View call graph

- Red for high consumers
- Green for low consumers

Demo

Plot memory usage with memory_profiler

View cProfile output

- Snakeviz
- PyCharm Professional

Module Summary

- Understanding performance
- Strategy to improve performance
- Basic ways to measure performance
- Why profile?
- More profilers
- Visualizing profiling data

Course Overview

Using the right data structures

Optimizing Python code

Using more threads

Using asynchronous code

Using more processes

Up Next:

Using the Right Data Structures