

# Faster Python Programs - Measure, don't Guess

A Tutorial at PyCon 2017

May 18, 2017

Portland, OR, USA

**author:** Dr.-Ing. Mike Müller

**email:** [mmueller@python-academy.de](mailto:mmueller@python-academy.de)

**twitter:** @pyacademy

**version:** 3.5

## About Me

- Enthusiastic Python user since 1999
- Python trainer since 2004
- CEO of Python Academy
- Chair of EuroSciPy 2008, 2009, PyCon DE 2011, 2012 and EuroPython 2014
- Chair of PySV e.V. (German equivalent to the PSF)

# About You

- How many years of Python?
- Python 2 or 3?
- Do you do number crunching?
- Are you problems CPU bound?
- Or memory bound?
- Or IO bound?

# Overview

- Hands-on sessions
- IPython Notebook recommended (standard Python and editor works too)
- PDF handout
- Download of examples
- Exercises after each topics
- 20 minutes break
- Please give feedback via Guidebook or Web (<https://www.surveymonkey.com/r/WVVS7R9>)

# How Fast is Fast Enough?

- Python is very high-level
- Slow for some types of computations
- This may or may not be a problem
- Before starting to optimize consider the costs
- Optimized code may need more effort to develop and maintain
- Balance between speed of development and speed of program execution

# Optimization Guidelines

Premature optimization is the root of all evil.

D. Knuth or C. A. R. Hoare or folklore (attributions seems not totally clear)

# Correct First - Faster Later

- Make sure your program works correctly
- Before you start thinking about optimization
- Never optimize before the program produces the desired results

# The Prize of Optimization

- Optimization often comes with a price
- It tends to make your code less readable
- More maintenance effort
- Readability and maintainability can suffer
- Python's excellent readability is not for free in terms of runtime
- You cannot always solve this contradiction between both goals



# General Guidelines

- ... for optimization

# 1. Make sure your program is really too slow

- Do you really need more performance?
- Are there any other slowdown factors such as network traffic or user input that have more impact on speed?
- Does it hurt if the program runs slowly?

## 2. Don't optimize as you go

- Don't waste time before you are certain that you will need the additional speed.

### **3. Only realistic use cases and user experience should be considered**

- Maximal expected number of users
- How often the program is used

## 4. Search for external causes for slowdown

- Network connections
- Database access
- Calls to system functions

## 5. Architecture can be essential for performance

- Is it appropriate?

## 6. Are there any bugs that slow down the program?

- Don't optimize with bugs
- The bug might be the bottleneck

## **7. If the program is really too slow, find the bottlenecks by profiling**

- That is what this tutorial is about



## 8. Always check the result of optimization

- Use unit tests
- With high coverage

## 9. Go for algorithms first

- Typically most performance gains come from better algorithms
- Find the big-O notation of your algorithm

## 10. Throw better hardware on it

- Often hardware is cheaper than programmer time
- You often easily add memory (avoid swapping)
- Use the fastest processor you can get