# Effective Programming Practices for Economists

# Scientific Computing

## Introduction to making code fast

Janoś Gabler and Hans-Martin von Gaudecker

# What do we mean by speedup

- Same calculations
- Same language
- Faster execution

# Speed can vary within a language

```
>>> def my_sum(numbers):
...     out = 0
...     for number in numbers:
...         out += number
...     return out

>>> numbers = list(range(10_000))

>>> %timeit my_sum(numbers)

128 μs ± 1.65 μs

>>> %timeit sum(numbers)

28.5 μs ± 275 ns
```

- In this simple example, the speed difference is 4.5x
- Speed differences of 100x are common, more is possible
- It gets really slow if you do not use libraries as intended

# Python can be really fast

- Numba uses the same technology as Julia (llvm)
- JAX uses technologies Julia dreams of and is even developing them further
- State of the art AI is trained in Python
- We have beat Fortran code with Python code several times

# Only optimize bottlenecks

> We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil (Donald Knuth)

- Typically, runtime is concentrated in a few sections of code
- Making the rest faster will not change overall runtime
- Important: Learn how to find those sections!

# Process

> If it doesn’t work, it doesn’t matter how fast it doesn’t work (Mich Ravera)

- Get it to run
- Get it right
- Find the bottleneck
- Speed up the bottleneck on one core
- Think about parallelization
- Repeat