Profiling and Optimizing Python Code

Cameron Hummels
NSF Fellow, Caltech

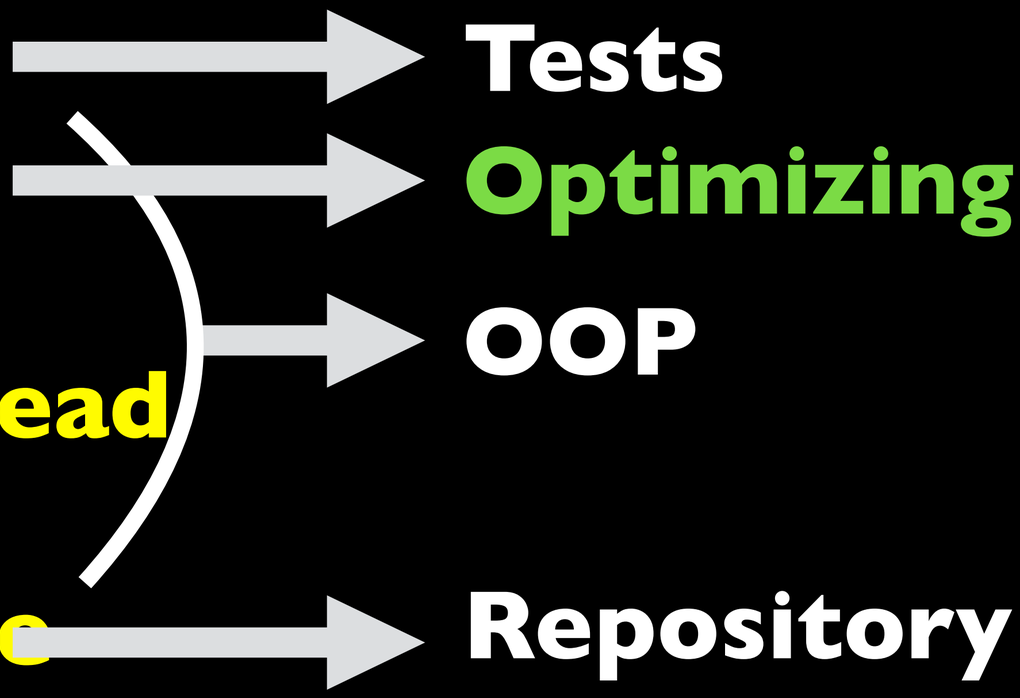
Code should* be:

Code should* be:

- **correct**
- **efficient**
- **simple**
- **easy to read**
- **concise**
- **accessible**

Code should* be:

- correct
- efficient
- simple
- easy to read
- concise
- accessible



Optimizing Code: what not to do

"Premature optimization is the root of all evil."
—Donald Knuth

Optimizing Code: what are you optimizing?

- CPU speed
- System memory
- Input/Output
- Storage Space

Profiling utilities

- time — coarse total time measurement
- timeit — good for testing code snippets
- cProfile — deep testing of total program
  - pstats — text-based viz for cProfile outputs
  - snakeviz — viz tool for cProfile outputs
  - runsnakerun — pipeline tool for cProfile out
  - pyprof2html — html tool for cProfile outputs
- line_profiler — line-by-line profiling in one fun
- memory_profiler — line-by-line memory profile

Live-coding Example

Concepts to remember

- Decide what you are optimizing over
- Computer time versus person time
- Write readable code first, then optimize
- Use profilers to identify bottlenecks
- Address bottlenecks one at a time
- Latest Python is most optimized
- Try new approaches and profile/testit
- Object oriented code

Concepts to remember

- time for coarse level profiling
- timeit for quick profiling
- cProfile for deep profiling with lots of viz tools
- line_profiler (kernprof) for line-by-line functions
- memory_profiler for memory consumption and memory leaks

Concepts to remember

- NumPy arrays are optimized
- Vectorize loops when possible
- List comprehensions
- Avoid building lists through appends
- In place operations as opposed to rebuilding
- Cython
- Numba
- Algorithm design and parallelization