APC 524/AST 506/MAE 506

Software Engineering for Scientific Computing

Instructors

Henry Schreiner Office TBD henryfs@princeton.edu Office hours: TBD	Romain Teyssier 134 Peyton Hall teyssier@princeton.edu Office hours: TBD

Schedule

Tuesday and Thursday, 3:00 pm to 4:20 pm

Overview

The goal of this course is to teach basic tools and principles of writing good code, in the context of scientific computing. Specific topics include an overview of relevant compiled and interpreted languages, build tools and source managers, design patterns, design of interfaces, debugging and testing, profiling and improving performance, portability, and an introduction to parallel computing in both shared memory and distributed memory environments. The focus is on writing code that is easy to maintain and share with others. You will develop these skills through a series of programming assignments and a group project.

Syllabus

A rough outline of the topics to be covered is given below. This is not meant to be a week-by-week outline, but rather an indication of the topics to be covered.

1. Introduction and motivation
    a. The importance of software engineering in scientific computing
    b. Problem-solution ordering
    c. Unix setup and tools
    d. Version control overview

    e. Language overview
2. Testing
    a. Introduction to pytest
    b. Test driven development
    c. Parametrising
    d. Fixtures
    e. Debugging
3. Object oriented programming
    a. Code reuse
    b. Separation of concerns
    c. Introduction to classes
    d. Overloading
    e. Inheritance
4. Design patterns
    a. Mutability and state
    b. Functional programming
    c. Memory safety
    d. Modularity
    e. Refactoring
5. Static typing
    a. Running a type checker (similar to compile step)
    b. Type narrowing
    c. Protocols (& interfaces)
    d. Generics
6. Version Control (Git & GitHub)
    a. Intro to git
    b. Investigating history
    c. Tagging and branching
    d. Manipulating history
    e. Collaborative development
    f. Hooks and configuration
7. Packaging and quality control
    a. Introduction to Python Packaging and build systems
    b. Setting up pre-commit
    c. Useful pre-commit hooks
    d. Intro to task runners (nox)
    e. Conda/Mamba/MicroMamba and conda-forge
8. Continuous integration
    a. Introduction to GitHub Actions
    b. Setting up continuous testing
    c. Setting up deployment
    d. Setting up other systems (like pre-commit.ci)
9. Intro to a compiled language
    a. Intro to C++/Fortran

    b. Intro to compilers and linkers
    c. Intro to C++ build systems (like CMake)
    d. Intro to C++ tooling
    e. Setting up a compiled project
    f. Templates and compile time programming
10. Profiling
    a. Using a Python profilers
    b. Using a profiler with a compiled language
    c. Memory optimization
11. Parallel computing
    a. Threaded programming in Python
    b. Multiprocessing in Python
    c. Using OpenMP in C++
    d. MPI basics
    e. Running on nodes for HPC
12. Mixing languages
    a. Using a shared object library
    b. Bindings for Python
    c. Calling Python from a C++
13. Exotic topics
    a. WebAssembly
    b. Useful shell tools and tricks

Communication

This course will use Canvas.

Grading

The grades for the course come from 5-8 programming assignments (50%) and a final project (50%). The final project will take about 2ish months during the later portion of the course. Groups of 3-5 students will work on projects.

- Design plan
- Documentation
- Version control
- Automated testing
- Some sort of profiling with some iterative performance improvement

Individuals will:

- Take part in 1-2 design and code reviews
- 10-15 pages of written description, plus 2K-4K LoC (rough estimate)

- Public presentation (15-20 minutes per group)