

Computing Methods for Experimental Physics and Data Analysis

Introduction

L. Baldini, G. Lamanna, A. Manfreda, A. Retico, A. Rizzi

luca.baldini@pi.infn.it

Università and INFN–Pisa

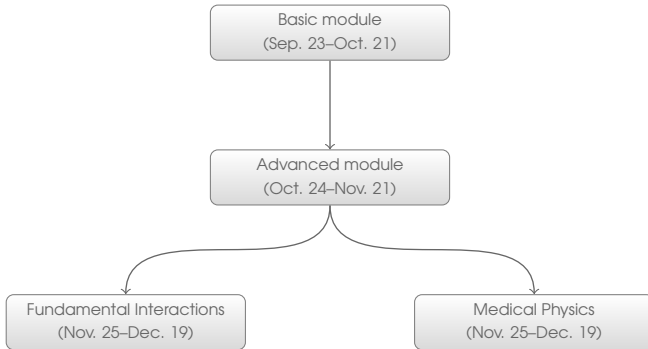
October 17, 2019



Goals and prerequisites

- ▷ What is this all about?
 - ▷ Automating repetitive tasks
 - ▷ Python basics, standard library and scientific ecosystem
 - ▷ Collaborative code development and best practices
 - ▷ Algorithms and data structures
 - ▷ Machine learning
 - ▷ Specific tools for high-energy physics or medical physics
- ▷ This is not so much about Python or C++—it is about how to write code for effective data analysis
- ▷ Will I be a professional data scientist at the end of the semester?
 - ▷ No, but hopefully you'll be able to poke around and find the right tool for the job at hand
- ▷ What do I need to know in advance?
 - ▷ Have a vague idea of how a computer operates
 - ▷ If you have ever programmed before that would be great!
- ▷ This is our first round: we will adjust along the way

Basic structure of the course



▷ Modularity and standard paths:

- ▷ Each module is worth 3 credits
- ▷ 6 credits: basic + advanced
- ▷ 9 credits: basic + advanced + fundamental interactions
- ▷ 9 credits: basic + advanced + medical physics



Basic module

L. Baldini and A. Manfreda

- ▷ Collaborative tools
 - ▷ Version control, development workflow, development platforms
- ▷ Python basics
 - ▷ Coding conventions, structuring a package
 - ▷ Variables, native types, functions
 - ▷ The Python standard library
- ▷ Algorithms and data structures
 - ▷ Complexity and asymptotic running time
 - ▷ Python data structures and native algorithms
- ▷ Object-Oriented Programming (OOP)
 - ▷ Classes, inheritance, composition
 - ▷ Operator overload and emulation of Python builtin types
- ▷ The Python computing ecosystem
 - ▷ numpy: arrays, functions, broadcasting
 - ▷ Vectorization
 - ▷ Scipy: plotting and fitting
 - ▷ Pandas



Advanced module

L. Baldini, G. Lamanna, A. Manfreda, A. Rizzi

- ▷ Advanced code development
 - ▷ Unit testing, continuous integration, static analysis, documentation
- ▷ Advanced Python
 - ▷ Errors, exceptions, iterators and generators, decorators
 - ▷ Profiling and optimization
- ▷ Parallel computing
 - ▷ Computer architectures, memory, scaling laws, CPUs and GPUs
 - ▷ Parallel programming: concurrency and parallelism, threading in Python
- ▷ Machine learning
 - ▷ Classification and regression: boosted decision trees and multilayer perceptrons
 - ▷ Deep learning: neural networks, the keras library
 - ▷ Supervised and unsupervised training, reinforcement learning
 - ▷ Tensorflow



Fundamental Interactions

G. Lamanna, A. Rizzi

- ▷ Introduction to C++
 - ▷ Coding style and organization, declaration of interfaces
 - ▷ Classes: constructors, virtual functions, private and public, abstract classes, inheritance
 - ▷ References, pointers, dynamic memory allocation, memory ownership, smart pointers
 - ▷ Templates, standard template library
 - ▷ C++11 and C++14: lambda functions, auto variables
- ▷ More parallel computing
 - ▷ Cuda and OpenCL
 - ▷ Examples of algorithms for HEP
 - ▷ GPU in HEP Data Analysis
- ▷ The ROOT data analysis framework
 - ▷ ROOT toolkit
 - ▷ PyROOT, root-numpy, RDataFrame



- ▷ Medical data processing and feature extraction (python/MATLAB)
 - ▷ Tools for handling standard-format medical data (DICOM)
 - ▷ Data anonymization and visualization
 - ▷ Deriving features from images, image segmentation
 - ▷ Data quality control pipelines: outlier removal, dimensionality reduction
- ▷ Data analysis and classification (python/MATLAB)
 - ▷ Performance evaluations: figures of merit, cross-validation schemes, permutation test
 - ▷ Machine-learning and deep-learning tools for segmentation and classification
 - ▷ Data augmentation, transfer learning, retrieving localization information.



Logistics

Timetable and final exam

- ▷ Timetable
 - ▷ Monday, 09:00–11:00 (room G1)
 - ▷ Monday, 16:00–18:00 (room H)
 - ▷ Thursday, 09:00–11:00 (room M)
- ▷ Lectures on Monday morning and Thursday morning (2×2 hours)
 - ▷ Tentative idea: 1 hour with slides and 1 hour at the computer
 - ▷ (Actual mileage might vary)
- ▷ “Lab” on Monday afternoon
 - ▷ Attack a small project each week in a 2–3 hour session
 - ▷ (Bring your own laptop!)
- ▷ **And, of course, the final exam**
 - ▷ Development of a specific, reasonable-size software project (related to the topics covered in the course)
 - ▷ Two-page description of the project and source code made available in advance
 - ▷ Oral exam (starting from the discussion of the project)