Cloud Contraptions
TRAINING & CONSULTING

To discuss this course and customizations:
Call: +1 434-509-6890 or Email: sales@cloudcontraptions.com

High-Performance Python with C and C++

Class Duration

7 hours of live training delivered over 1 day.

Student Prerequisites

- All students should have significant experience with Python
- Working knowledge of C, and for the PyBind11 and nanobind sections, basic familiarity with C++
- The class has no review of the Python, C, or C++ programming languages
- If you need to learn C programming, please consider the C Programming for Python Developers course

Target Audience

This course is for Python developers who need to accelerate Python code by reaching into native code. Students should be comfortable writing production Python and have enough C (and ideally C++) background to follow native-code examples, build steps, and debugging sessions.

Description

This one-day course is for Python developers who need to accelerate Python code by reaching into native code. The course surveys the practical options for high-performance Python - Numba for JIT-compiling NumPy-heavy code, CFFI for wrapping existing C libraries, modern C++ bindings with PyBind11 and its successor nanobind, and the Python C API for direct C extensions - and gives students the judgment to pick the right tool for each problem. The course closes with a focused discussion of the GIL, releasing the GIL in native code, and free-threaded Python (PEP 703), which became officially supported in Python 3.14 (PEP 779) and is now changing how extension authors think about thread safety.

Learning Outcomes

- Understand how and why Python is slow for CPU-bound workloads

To discuss this course and customizations:
Call: +1 434-509-6890 or Email: sales@cloudcontraptions.com

- Profile Python applications to identify hot spots worth optimizing
- Accelerate NumPy-heavy Python code with Numba
- Wrap existing C libraries with CFFI in both ABI and API modes
- Build C and C++ extensions with PyBind11 and nanobind, and choose between them
- Write direct C extensions using the Python C API and package them with Stable ABI for cross-version compatibility
- Reason about reference counting, memory management, and the GIL in native code
- Understand the implications of officially-supported free-threaded Python (PEP 703 / PEP 779) for extension authors
- Compare the approaches and choose the right tool for each problem

Training Materials

All students receive comprehensive courseware covering all topics in the course. Courseware is distributed via GitHub in the form of documentation and extensive code samples. Students practice the topics covered through challenging hands-on lab exercises.

Software Requirements

Students will need a free, personal GitHub account to access the courseware. Students will need permission to install Python, a C and C++ compiler toolchain, and Visual Studio Code on their computers. Also, students will need permission to install Python Packages and Visual Studio Code extensions. If students are unable to configure a local environment, a cloud-based environment can be provided.

Training Topics

Why Python is Slow and Decision Framework

- Why Python is Slow for CPU-Bound Workloads
- CPU-Bound vs IO-Bound Workloads
- Profiling Python Applications with cProfile and py-spy
- Decision Framework: Numba, CFFI, PyBind11, nanobind, C Extensions, or Cython

Numba

- What is Numba?

- @jit and @njit Decorators
- Type Signatures and Specialization
- NumPy Interop with Numba
- Numba on Free-Threaded Python (Experimental in 0.63; Python 3.14t Support in 0.65+)
- Where Numba Shines and Where It Doesn't
- Common Pitfalls and Limitations

CFFI

- What is CFFI?
- ABI Mode vs API Mode
- Compile C Code to a Shared Object
- Wrapping Existing C Libraries
- C Types, Structs, and Arrays in CFFI
- Callbacks from C to Python
- Error Handling and Memory Ownership

Modern C++ Bindings: PyBind11 and nanobind

- PyBind11 and nanobind: Two Closely-Related Tools (nanobind by PyBind11's Original Author as a Newer Alternative)
- When to Choose nanobind over PyBind11 (Compile Time, Binary Size, Runtime Overhead, Stable ABI)
- Exposing C and C++ Functions to Python
- Exposing C++ Classes with Constructors, Methods, and Properties
- STL Container Conversions (vector, map, set)
- Smart Pointers and Object Lifetime
- NumPy Interop: py::array_t (PyBind11) and nanobind's Multi-Framework ndarray (NumPy, PyTorch, JAX, TensorFlow)
- Building with setuptools, scikit-build-core, and CMake
- Stable ABI Builds with nanobind (Python 3.12+; Note: STABLE_ABI Is Ignored Under FREE_THREADED Until PEP 803 Lands)
- Packaging Extensions for PyPI

High Performance with C: The Python C API

- When to Drop Down to the Raw Python C API
- Basic Structure of a C Extension
- PyObject, Argument Parsing, and Return Values
- Managing Memory and Reference Counting

To discuss this course and customizations:
Call: +1 434-509-6890 or Email: sales@cloudcontraptions.com

- NumPy C API: Passing and Creating Arrays
- Limited API and Stable ABI: abi3 Wheels Today (PEP 384), PEP 803 (abi3t, Accepted Targeting 3.15), PEP 809 (abi2026, Draft - Proposed Future Direction)
- HPy: A Portable Alternative C API (Brief Overview)
- Debugging C Extensions and Detecting Memory Leaks
- Packaging C Extensions for PyPI

GIL, Concurrency, and Free-Threaded Python

- What the GIL Is and Why It Exists
- Implications for C, C++, and Numba Extensions
- Releasing the GIL in Native Code (Py_BEGIN_ALLOW_THREADS, PyBind11's py::gil_scoped_release, nanobind's nb::gil_scoped_release)
- Free-Threaded Python (PEP 703): What Changes and What Doesn't
- PEP 779: Free-Threaded Python is Officially Supported in 3.14 (No Longer Experimental)
- Declaring Free-Thread Safety in Extensions (Py_mod_gil, PyBind11 py::mod_gil_not_used, nanobind FREE_THREADED Flag)
- Implications for Extension Authors and Library Maintainers
- Thread-Safety Considerations Going Forward

