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, C++, and Cython

Class Duration

21 hours of live training delivered over 3 days.

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 C Programming for Python Programmers first

Target Audience

Python developers who need to speed up CPU-bound code with native extensions, and library maintainers who build and ship them. Teams that prefer Rust as the native layer can take High-Performance Python with Rust instead.

Description

This three-day course covers the main ways to speed up Python with native code and how to choose between them. It starts with profiling, including the Tachyon sampling profiler, and with measuring the cost of crossing into native code. Day one covers ctypes, CFFI, and the Python C API, including stable ABI wheels with abi3 and abi3t. Day two covers C++ bindings with pybind11 and nanobind, built with scikit-build-core and CMake, and starts Cython. Day three covers typed memoryviews, wrapping C libraries, parallel loops, free-threaded extensions, and publishing wheels with cibuildwheel.

Learning Outcomes

- Profile Python code to find hot spots worth moving to native code
- Call existing C libraries with ctypes and CFFI
- Write C extensions with the Python C API

Copyright 2026 Cloud Contraptions LLC – All Rights Reserved

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

- Manage references, errors, and module state in C extensions
- Build stable ABI wheels with abi3 and abi3t
- Bind C++ code with pybind11 and nanobind
- Compile and type Python code with Cython
- Optimize numerical loops with typed memoryviews
- Wrap C libraries with Cython
- Release the GIL safely in native code
- Prepare C, C++, and Cython extensions for free-threaded Python
- Build and publish cross-platform wheels with cibuildwheel

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 (including the free-threaded build), a C and C++ compiler toolchain, CMake, and Visual Studio Code on their computers. Also, students will need permission to install Python Packages and Visual Studio Code extensions. The course is taught on Python 3.15, Cython 3.3, pybind11 3.1, and nanobind 3.1. If students are unable to configure a local environment, a cloud-based environment can be provided.

Training Topics

Why Python is Slow and Choosing a Target

- Why Python is Slow for CPU-Bound Workloads
- CPU-Bound vs. IO-Bound Workloads
- The Tachyon Sampling Profiler
- `profiling.tracing` and py-spy
- Honest Benchmarks
- Warmup, Variance, and Realistic Data Sizes
- The Cost of Crossing into Native Code
- Choosing a Native-Code Approach

Copyright 2026 Cloud Contraptions LLC – All Rights Reserved

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

Calling C Libraries with ctypes and CFFI

- Loading Shared Libraries with ctypes
- CFFI ABI Mode vs. API Mode
- Compiling C Code to a Shared Object
- C Types, Structs, and Arrays
- Callbacks from C to Python
- Error Handling and Memory Ownership

The Python C API

- When to Drop Down to the Raw C API
- Extension Module Structure
- Multi-Phase Initialization and Module State
- `PyObject`, Argument Parsing, and Return Values
- Owned and Borrowed References
- Raising and Propagating Exceptions from C
- Defining Extension Types
- The Buffer Protocol and the NumPy C API
- Debugging with Debug Builds and Sanitizers
- Leak Detection

The Limited API and Stable ABI

- Limited API vs. Stable ABI
- abi3 Wheels Across CPython Versions
- abi3t for Free-Threaded Builds
- The `Py_TARGET_ABI3T` Macro
- The Module Export and Slot APIs
- Choosing Between abi3, abi3t, and Version-Specific Wheels
- Costs of the Limited API

Modern C++ Bindings: pybind11 and nanobind

- Shared Design, Different Trade-Offs
- Compile Time and Binary Size
- Runtime Overhead and Feature Coverage
- Exposing C and C++ Functions to Python
- Exposing C++ Classes
- STL Container Conversions
- Smart Pointers and Object Lifetime

Copyright 2026 Cloud Contraptions LLC – All Rights Reserved

- pybind11's `smart_holder`
- Mapping C++ Enums to Native Python Enums
- NumPy Arrays with `py::array_t`
- nanobind's `ndarray` for NumPy, PyTorch, and JAX
- Building with scikit-build-core and CMake

Cython Fundamentals

- What is Cython?
- When to Use Cython
- Compiling Pure Python Code with Cython
- Building with setuptools or meson-python
- Inspecting Generated C Code with HTML Annotation
- Quick Compilation with Jupyter `%%cython`
- `def`, `cdef`, and `cpdef` Functions
- `@cython.cfunc` and `@cython.ccall`
- Pure Python Mode with Type Hints
- Classic Cython Syntax
- Language Level and `noexcept`
- Exception Propagation
- `.pyx` and `.pxd` Files
- Sharing Declarations Across Modules

Optimizing Numerical Code with Cython

- Compiler Directives such as `boundscheck`
- Typed Memoryviews
- Passing NumPy Arrays to Cython
- NumPy 2 Compatibility
- Fused Types
- Performance Comparison with NumPy and Numba

Wrapping C Libraries with Cython

- Declaring External C Code
- C Structs, Unions, Enums, and Typedefs
- Error Handling Across the Python and C Boundary
- C Callbacks Calling Back into Python
- Memory Ownership and Resource Management
- Extension Types with `cdef class`

Copyright 2026 Cloud Contraptions LLC – All Rights Reserved

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

- Special Methods and the Buffer Protocol

The GIL, Parallelism, and Free-Threaded Python

- The GIL and Native Code
- Releasing the GIL with `Py_BEGIN_ALLOW_THREADS`
- `gil_scoped_release` in pybind11 and nanobind
- Cython `nogil` Blocks
- Parallel Loops with Cython `prange` and OpenMP
- Free-Threading for Extension Authors
- Declaring Free-Thread Safety
- `Py_mod_gil` and `py::mod_gil_not_used`
- nanobind's `FREE_THREADED` Flag
- Cython's `freethreading_compatible` Directive
- Critical Sections with `PyCriticalSection`
- `py::scoped_critical_section`
- `cython.critical_section` and `cython.pymutex`
- Cython's Experimental Free-Threading Support
- Module-Level State and Thread Safety

Building and Distributing Extensions

- setuptools, scikit-build-core, and meson-python
- Cross-Platform Wheels with cibuildwheel
- Building Free-Threaded and Stable ABI Wheels
- Packaging Extensions for PyPI and Conda
- Publishing from CI with Trusted Publishing

Copyright 2026 Cloud Contraptions LLC – All Rights Reserved