📘 Full Session Material – As Text

# 🧪 Cython & pybind11

Performance Optimization in Python

Teaching Guide for Intermediate Python Developers

🔍 Overview

This guide teaches students how to:

- Optimize performance-critical Python modules using Cython

- Integrate C++ logic into Python using pybind11

- Measure and benchmark performance improvements

🧱 Section 1: Introduction to Cython

What is Cython?

- A superset of Python that supports static typing and compilation to C

- Allows writing Python-like code that compiles to high-performance C extensions

Why Use It?

- Speed up CPU-bound operations (e.g., loops, math)

- Interface with C/C++ libraries

- Keep most of your code in Python while optimizing critical parts

⚙️ Section 2: Writing Cython Versions of Critical Modules

Step-by-step Setup

Install Dependencies

pip install cython numpy

Create Your Project Structure

cython\_demo/

├── performance\_benchmark.py # Pure Python version

├── fib\_cython.pyx # Cython version

├── setup.py # Build script

└── benchmark\_fib.py # Benchmark script

# Example: Fibonacci Sequence (Python vs Cython)

`performance\_benchmark.py`

python

def fib\_python(n):

a, b = 0, 1

result = []

while a < n:

result.append(a)

a, b = b, a + b

return result

`fib\_cython.pyx`

cython

def fib\_cython(int n):

cdef int a = 0, b = 1, temp

result = []

while a < n:

result.append(a)

temp = a

a = b

b = temp + b

return result

`setup.py`

python

from setuptools import setup

from Cython.Build import cythonize

setup(

name='Cython Demo',

ext\_modules=cythonize("fib\_cython.pyx"),

)

`benchmark\_fib.py`

python

import timeit

from performance\_benchmark import fib\_python

import fib\_cython

def benchmark():

n = 1\_000\_000

print(f"Running Fibonacci up to {n}...\n")

py\_time = timeit.timeit(lambda: fib\_python(n), number=10)

cy\_time = timeit.timeit(lambda: fib\_cython.fib\_cython(n), number=10)

print(f"Python version: {py\_time:.4f}s")

print(f"Cython version: {cy\_time:.4f}s")

print(f"Speedup: {py\_time / cy\_time:.2f}x")

if \_\_name\_\_ == "\_\_main\_\_":

benchmark()

# Using `pyximport` for Quick Testing

python

# test\_pyximport.py

import pyximport

pyximport.install()

from fib\_cython import fib\_cython

print(fib\_cython(100))

> Note: This won’t work for functions that use `prange` or need parallelism.

# Typed Memoryviews for Large Data

`sum\_cython.pyx`

cython

def sum\_cython(int[:] a, int[:] b):

cdef int i, n = len(a)

result = [0] \* n

for i in range(n):

result[i] = a[i] + b[i]

return result

`benchmark\_sum.py`

python

import array

import timeit

import sum\_cython

def get\_data():

size = 1\_000\_000

a = array.array('i', range(size))

b = array.array('i', range(size))

return (a, b)

def sum\_python(a, b):

return [a[i] + b[i] for i in range(len(a))]

def benchmark():

args = get\_data()

py\_time = timeit.timeit(lambda: sum\_python(\*args), number=10)

cy\_time = timeit.timeit(lambda: sum\_cython.sum\_cython(\*args), number=10)

print(f"Python version: {py\_time:.4f}s")

print(f"Cython version: {cy\_time:.4f}s")

print(f"Speedup: {py\_time / cy\_time:.2f}x")

if \_\_name\_\_ == "\_\_main\_\_":

benchmark()

📊 Section 3: Benchmarking Cython vs Python

Use `timeit`, `matplotlib`, or export results to CSV for deeper analysis.

🔗 Section 4: Integrating C++ with Python using pybind11

# What is pybind11?

- Lightweight header-only library to expose C++ types in Python

- Works well with modern C++ (C++11+)

- Easy integration via `setuptools`

# Installation

pip install pybind11

Also install a compiler:

- Windows: Microsoft C++ Build Tools

- Linux/macOS: `g++` or `clang`

# Example: Distance Calculation in C++

`shapes.cpp`

cpp

#include <pybind11/pybind11.h>

#include <cmath>

namespace py = pybind11;

struct Point {

double x;

double y;

};

double distance(const Point& a, const Point& b) {

return std::sqrt((a.x - b.x)\*(a.x - b.y));

}

PYBIND11\_MODULE(shapes, m) {

py::class\_<Point>(m, "Point")

.def(py::init<double, double>())

.def\_readwrite("x", &Point::x)

.def\_readwrite("y", &Point::y);

m.def("distance", &distance, "Calculate distance between two points");

}

`setup\_shapes.py`

python

from setuptools import setup

from pybind11.setup\_helpers import Pybind11Extension, build\_ext

ext\_modules = [

Pybind11Extension("shapes", ["shapes.cpp"]),

]

setup(

name="shapes",

cmdclass={"build\_ext": build\_ext},

ext\_modules=ext\_modules,

)

`test\_shapes.py`

python

from shapes import Point, distance

p1 = Point(1.0, 2.0)

p2 = Point(4.0, 6.0)

print(f"Distance between {p1.x, p1.y} and {p2.x, p2.y}: {distance(p1, p2):.2f}")

🎯 Section 5: When to Use Cython vs pybind11

| Tool | When to Use | Pros | Cons |

||-|||

| Cython | Optimizing existing Python code | Fast to prototype, no C++ knowledge required | Limited to Python-like syntax |

| pybind11 | Wrapping C++ libraries or writing new logic in C++ | Full control over memory and speed | Requires C++ knowledge |

📝 Appendix A: Exercise Files for Students

See below for downloadable exercises.

🛠️ Appendix B: Common Errors and Fixes

| Error | Cause | Fix |

||-|--|

| `TypeError: a bytes-like object is required` | Passing list to typed memoryview | Use `array.array` or `numpy.ndarray` |

| `Cannot find cl.exe` | Missing Visual Studio tools | Install Microsoft C++ Build Tools |

| `ImportError: DLL load failed` | Compiler mismatch | Rebuild with correct Python/C++ versions |

📌 Appendix C: Compiler Setup Guides

# Windows

Install from: https://visualstudio.microsoft.com/visual-cpp-build-tools/

Select:

- "C++ build tools"

- "Windows SDK"

# macOS

xcode-select --install

# Linux (Ubuntu)

sudo apt install build-essential g++

# 📁 Student Exercises

Below are starter templates for student exercises.

🧪 Exercise 1: Optimize List Sum Function with Cython

# Goal:

Use Cython to speed up a list addition function.

# Steps:

1. Write a pure Python function `sum\_python(a, b)`

2. Create a `.pyx` file with a Cython version using typed memoryviews

3. Compile using `setup.py`

4. Benchmark both versions using `timeit`

🧪 Exercise 2: Build a Fibonacci Generator in Cython

# Goal:

Speed up the Fibonacci sequence generator using static typing.

# Steps:

1. Start with a pure Python implementation.

2. Convert to Cython by adding `cdef` variables.

3. Compile and benchmark.

4. Compare performance with large inputs (`n = 1\_000\_000`).

🔗 Exercise 3: Integrate a C++ Class into Python using pybind11

# Goal:

Expose a C++ class to Python using pybind11.

# Steps:

1. Define a `Rectangle` class in C++ with methods like `area()`, `perimeter()`.

2. Use pybind11 to expose the class.

3. Compile and import in Python.

4. Instantiate and call methods from Python.

📥 Final Files Delivered

All files have been provided here in full. You can now:

- Copy and paste into your own files

- Save as `.py`, `.cpp`, `.pyx`, etc.

- Zip them together for distribution