Python Setup & Installation

Start here if you've never installed Python. No prior setup assumed.

This page gets you a working Python environment for lab work: Python itself, an editor, the analysis packages everyone needs, and — only if your work needs them — the packages and drivers for talking to instruments. Many lab computers already have all of this; follow the steps below to set up your own machine.

1 What you're installing, and why

Native Python from python.org, with VS Code as the editor. Three reasons, in order of how much they matter:

- **Instrument drivers expect it.** NI-DAQmx, NI-488.2, and Thorlabs Kinesis look for a system Python they can bind to. If you ever drive hardware from Python, this is the install that works.
- **One Python, one package manager.** Native Python plus `pip` is a single environment you can reason about. Adding a second package ecosystem alongside `pip` is the most common way a working setup quietly stops working.
- **One editor for both halves of the job.** VS Code runs notebooks and scripts, so moving from exploring data to automating a measurement doesn't mean changing tools.

If you already have Anaconda, see If you already have Anaconda below rather than uninstalling it.

> **Note for Mac users**
>
> Instrument work needs Windows. None of the drivers below has a current macOS build:
>
> - **NI-DAQmx** (data acquisition) — Windows and Linux; no macOS build at all
> - **NI-488.2** (GPIB) — Windows and Linux; the last macOS build was 2022
> - **NI-VISA** (USB/Ethernet instruments) — Windows and Linux. A macOS build exists but is frozen at version 2023 Q3, supports only macOS 12, and dropped GPIB support — treat it as unavailable on a current Mac.
>
> On a Mac you can install Python and the core analysis packages (NumPy, SciPy, Matplotlib, and so on) and do all your analysis locally, but you'll need a Windows lab computer for data acquisition, GPIB, and motor control. For Ethernet, serial, or USB instruments specifically, pyvisa-py is a driver-free option that does work on a Mac.

2 Install Python (Windows)

Use the **Python Install Manager**, the official tool from python.org for managing Python versions on Windows. The two downloads below are identical builds; python.org suggests the Store for most people.

Option A: install from the Microsoft Store

Search for "Python Install Manager" in the Microsoft Store and install it.

Option B: download from python.org

1. Go to python.org/downloads
2. Click **"Download Python install manager"**

3. Run the installer and follow the prompts

> If `py list --online` errors about "legacy py.exe"
>
> An older Python launcher is still installed and shadowing the new one. Open **Settings** → **Apps** → **Installed Apps**, search for "Python Launcher", and uninstall it.

After installation, open PowerShell and install Python itself:

```
py install 3.13
```

Verify it:

```
py --version
```

To see the versions available:

```
py list --online
```

> Which version?
>
> This site targets **Python 3.13**. The scientific packages follow a support policy that drops older Python versions on a rolling schedule, and 3.12 falls out of that window at the end of 2026 — meaning it stops receiving new NumPy and SciPy releases. Newer versions than 3.13 work too, but 3.13 has had long enough for every package here to ship Windows builds for it.

The `py` launcher is how you'll invoke Python from here on — `py script.py` to run a script, `py -m pip ...` to install packages. It picks the right interpreter even with several versions installed.

3 Install Python (Mac)

1. Go to python.org/downloads
2. Download the **macOS installer** for the latest release (macOS 10.15 and later)
3. Run the installer and follow the prompts

Verify it by opening Terminal and running:

```
python3 --version
```

On Mac the command is `python3` (and `python3 -m pip`) wherever this site shows `py`.

4 Install VS Code

Download it from code.visualstudio.com and run the installer.

Then open the Extensions panel (`Ctrl+Shift+X`) and install two extensions:

Extension	Publisher	Purpose
Python	Microsoft	Python language support. Installing it also pulls in Pylance, Python Debugger, and Python Environments automatically
Jupyter	Microsoft	Run notebooks inside VS Code

With those in place, VS Code opens both `.py` scripts and `.ipynb` notebooks. Workflows covers when to reach for each and how to work in the editor.

Point VS Code at the right Python

Click the Python version in the bottom status bar to choose an interpreter. If code runs in the terminal but not in the editor — or an installed package "isn't found" — this is nearly always why.

5 Install the analysis packages

These are the packages every kind of lab analysis uses.

Windows:

```
py -m pip install numpy scipy matplotlib jupyterlab uncertainties
```

Mac:

```
python3 -m pip install numpy scipy matplotlib jupyterlab uncertainties
```

That covers everything under Analyzing Data — curve fitting, error propagation, Fourier analysis. Add the hardware packages below only if your work needs them.

Optional: virtual environments

The instructions above install packages for your user account, which is fine when everything you do shares one set of packages. If you work on several projects with conflicting requirements, give each its own virtual environment:

```
py -m venv .venv
.venv\Scripts\Activate.ps1
py -m pip install numpy matplotlib
```

VS Code detects a `.venv` in the project folder and offers to use it. Deactivate with `deactivate`.

If PowerShell refuses to run `Activate.ps1` — "running scripts is disabled on this system" — allow it once for your account:

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
```

This is worth doing for a long-lived project and unnecessary overhead for a one-off analysis.

6 Install hardware packages and drivers (Windows)

Talking to an instrument takes two pieces: a **Python package** (installed with `pip`) and a **vendor driver** (usually installed separately). They're largely independent — install only the pairs for hardware you actually use.

> **Downloading NI drivers**
>
> NI requires a free account to download drivers, so expect a sign-in prompt.

6.1 Bench instruments over VISA (USB / Ethernet / GPIB)

For oscilloscopes, function generators, power supplies, and multimeters. See VISA Instrument Control.

```
py -m pip install pyvisa
```

Then install the **NI-VISA driver**. Verify:

```
import pyvisa
rm = pyvisa.ResourceManager()
print(rm.list_resources())   # lists all connected instruments
```

An empty list means the driver is installed but sees no instruments — check that something is connected and powered on.

6.1.a No vendor driver: pyvisa-py

If you can't install NI-VISA — no admin rights, or a Mac — `pyvisa-py` is a pure-Python backend that needs no vendor driver:

```
py -m pip install pyvisa-py pyserial
```

```
import pyvisa
rm = pyvisa.ResourceManager('@py')   # '@py' selects the pure-Python backend
```

It handles Ethernet, serial, and USB instruments well. **GPIB is the exception** — it still needs a vendor GPIB driver underneath, so this is not a way around NI-488.2.

(`pyserial` is what pyvisa-py uses for serial ports. Note it installs as `pyserial` but imports as `serial`.)

6.2 GPIB instruments

For instruments on a GPIB (IEEE-488) bus behind an NI GPIB controller, also install the **NI-488.2 driver**. GPIB instruments then show up as VISA resources (`GPIB0::13::INSTR`, for example) and use the same `pyvisa` package — no extra Python package needed.

6.3 Data acquisition (NI DAQ devices)

For NI USB DAQ devices such as the USB-6009. See Data Acquisition (NI-DAQmx).

```
py -m pip install nidaqmx
```

Then install the NI-DAQmx driver. The package can fetch and run the installer for you, which is easier than finding it on NI's site:

```
py -m nidaqmx installdriver
```

Or download **NI-DAQmx** manually. Verify either way:

```
import nidaqmx
print(nidaqmx.system.System.local().driver_version)
```

6.4 Motorized stages (Thorlabs Kinesis)

For Thorlabs motorized stages and controllers. See Motor Control.

```
py -m pip install pythonnet
```

Then install **Thorlabs Kinesis**. Its .NET DLLs install to:

```
C:\Program Files\Thorlabs\Kinesis\
```

Code that drives a stage points at that directory, so note where it landed if you changed the install location.

> **Kinesis and XA**
>
> Thorlabs is gradually replacing Kinesis with a newer package, XA, which has an official Python interface and a longer support commitment. XA does not yet cover every controller, and Kinesis still has the broadest device support — so Kinesis remains the right choice for most hardware. If you're setting up a new controller, check whether XA supports it before committing to either.

7 Verify your setup

Run this to see what's working. Each hardware block is wrapped in `try/except`, so it reports cleanly whether or not that piece is installed — you should expect errors for hardware you didn't set up.

```
# Core analysis packages
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from uncertainties import ufloat
print("Core packages: OK")

# VISA (bench/GPIB instruments)
try:
    import pyvisa
    rm = pyvisa.ResourceManager()
    print(f"VISA resources found: {len(rm.list_resources())}")
except Exception as e:
```

```
    print(f"PyVISA: {e}")

# NI-DAQmx (data acquisition)
try:
    import nidaqmx
    print(f"NI-DAQmx version: {nidaqmx.system.System.local().driver_version}")
except Exception as e:
    print(f"NI-DAQmx: {e}")

# pythonnet (Thorlabs Kinesis)
try:
    import clr
    print("pythonnet: OK")
except Exception as e:
    print(f"pythonnet: {e}")

print("\nSetup complete.")
```

If the core packages import, you're ready for everything in Analyzing Data.

8 If you already have Anaconda

Anaconda works, and we don't recommend it. It bundles a second package manager (`conda`) alongside `pip`, and mixing the two is a reliable way to end up with an environment that breaks in confusing ways. If you're starting fresh, install native Python as above.

If Anaconda is already on your machine and you'd rather not change, here's what differs from the instructions on this site:

- **`py` points at the wrong Python.** Anaconda doesn't install the `py` launcher. If you have it from an earlier install, it resolves to python.org interpreters rather than your conda environment — which is the more confusing failure. Use `python` and `python -m pip` from an Anaconda Prompt or an activated environment.
- **Install into a named environment**, not `base`. Create one with `conda create -n lab python=3.13`, then `conda activate lab`.
- **`conda install` is fine for the analysis packages** — NumPy, SciPy and Matplotlib are all in the default channels and conda resolves them together.
- **The hardware packages aren't in the default channels**, but `pyvisa`, `pyserial`, `nidaqmx-python` and `pythonnet` are all on **conda-forge** (`conda install -c conda-forge pyvisa nidaqmx-python`). The conda builds tend to lag the PyPI releases, and you install the vendor driver separately either way — so this works, it just runs behind.
- **Tell VS Code which interpreter to use** — click the Python version in the status bar and pick the conda environment, not `base`.

9 Working without installing anything

Google Colab runs notebooks in your browser with nothing to install, and is a reasonable place to do analysis — particularly on a machine where you can't install software.

Its default runtime is on Google's servers, so it cannot reach instruments: no DAQ, no VISA, no motor control. Collect data on a lab computer, then upload the file to Colab to analyze it. Some courses use Colab as their main analysis environment; if yours does, follow that course's instructions.

(Colab can be pointed at a *local runtime* on your own machine, which does reach local hardware — but that needs a working local Python install anyway, and if you have that, you may as well use VS Code.)

Back to Python for Lab Work