ComfyLAB: Quick Start Guide & Comprehensive Overview

Comfortable Lab Automation Blocks — User Manual, Software Overview & Practical Reference

1. Introduction & Philosophy

1.1 What is ComfyLAB?

ComfyLAB (**Comf**ortable **L**ab **A**utomation **B**locks) is an open-source, visual programming platform engineered specifically for scientific experiment automation, test & measurement, data acquisition (DAQ), and real-time visualization.

In modern research laboratories, automated measurement setups often require interfacing with diverse hardware (oscilloscopes, power supplies, multimeters, function generators, custom sensors) via various protocols (VISA, SCPI, Serial RS-232/RS-485, Ethernet, native C/C++ DLLs). Writing raw script-based automation in traditional languages frequently results in verbose, repetitive boilerplate code for loop management, thread safety, exception handling, data parsing, and GUI rendering.

ComfyLAB resolves this by providing a clean, modular graphical interface where users drag functional **Blocks** onto a digital canvas, connect them with visual **Wires**, and organize results into an instant, interactive **Dashboard**.

1.2 Dual Architecture: Web UI + Python Server Engine

ComfyLAB operates on a decoupled client-server model separating user interaction from backend execution and hardware communication:

Layer	Primary Role	Key Capabilities	Stack
Front-End UI	User Interaction	• Infinite canvas with drag-and-drop wiring • Dedicated Operator Dashboard cockpit (`D`) • Real-time 2D/3D plots & whiteboard overlay	React, XY Flow, Plotly
API Layer	Data Transport	• Bidirectional WebSocket telemetry streaming • REST endpoints for execution & diagnostics	WebSockets, FastAPI
Backend Server	Execution & Hardware	• Push/pull hybrid state machine executor • VISA & Serial port lock manager • Virtual Instruments simulation server (SCPI) • Signal processing, NumPy/SciPy math & scripts	Python, PyVISA, NumPy, SciPy

- **Front-End Canvas & Dashboard**: Runs in modern web browsers (or single-file Desktop app builds). Provides fluid graphical manipulation, auto-layout tools, interactive graph zoom/pan, a digital whiteboard layer, and a dedicated **Operator Dashboard** that displays pinned cards, controls, and monitors in a streamlined control-room layout.
- **Backend Server**: Powered by Python and FastAPI. Handles real-time hardware locks, VISA/SCPI queries, heavy matrix calculations, multi-threaded execution, file persistence, Python virtual environment management, and an automated background simulation server for **Virtual Instruments**.

1.3 Built-in Safety & Security

- **Instrument Protection & Safety Teardown**: Physical laboratory hardware (lasers, high-voltage power supplies, RF signal sources) requires strict safety management. If a measurement run encounters an error, is manually stopped, or experiences a connection loss, ComfyLAB automatically executes safe shutdown hooks to turn off active outputs and restore instruments to safe states.
- **Blueprint Security Verification**: When opening blueprints from external sources, ComfyLAB runs security checks to prevent unauthorized script execution, giving users full control and prompt verification before running custom code.
- **Secure Remote Access**: ComfyLAB supports secure remote operation over networks using token authentication, allowing researchers to safely monitor and control experiments running on lab computers from remote workstations.
- **In-App Version & Update Checker**: Built-in release manager in the *About* dialog automatically checks for updates on PyPI / GitHub, displaying release notes and allowing one-click upgrades.

2. Capabilities & Flexibility: What Can (and Cannot) Be Done

ComfyLAB was built from the ground up to offer extreme flexibility without restricting advanced users to fixed GUI templates.

2.1 What Can Be Done with ComfyLAB

Feature Area	Capabilities & Highlights
Hardware Control	Full VISA support (GPIB, USB-TMC, TCP/IP, Serial RS-232/485) and SCPI. Built-in drivers for oscilloscopes, DMMs, power supplies, spectrometers.
Virtual Instruments	Built-in simulated oscilloscope (`virt_osc`), signal generator (`virt_siggen`), and RC circuit. Full offline testing without physical hardware.
Operator Dashboard	Dedicated cockpit view (hotkey `D`). Pin any block, control, or plot into organized Control and Monitor cards.
Progress & Timing	Countdown wait block (with skip button), progress bar widgets (0–100%), ETR clocks, and smart loops with `%` / ETR outputs.
Scientific Plots	Real-time 2D & 3D: XY line (linear/log), Dual-Y, Bar, Box plot, Histogram, 3D Surface/Scatter, Polar, Waterfall, and 2D Heatmaps.
Native Libraries	Direct C/C++ library calling (`.dll` on Windows, `.so` on Linux/macOS) via interactive Signature Editor without Python wrappers.
Polyglot Scripting	Inline custom blocks in 9 languages: Python, Rust, JS, TS, Julia, R, Lua, Octave, Wolfram. Access to `COMFYLAB_WORKSPACE` .
Python Ecosystem	Seamless integration with virtual environments (`.venv`), allowing any PyPI package (`scipy` , `pandas` , `torch` , `opencv`).
Packaging & Export	Publish custom blocks directly to the sidebar palette or bundle complete workflows into redistributable `.cfy` packages.
Signal Processing	NumPy matrix math, FFT power spectrum, curve fitting (Gaussian, exponential, polynomial), digital filtering, and peak detection.
Sub-Canvases	Group complex sub-graphs into clean, reusable Cluster blocks with dynamic I/O pins.
Whiteboard Layer	Draw freehand annotations, place notes, and draw geometric grouping boxes directly on the canvas (hotkey `4`).

2.2 Realistic Limitations: What Cannot (or Should Not) Be Done

While ComfyLAB handles almost any scientific measurement, automation, and data processing task, it is helpful to recognize its domain boundary:

- **Hard Real-Time Microsecond Determinism**: ComfyLAB runs on standard OS kernels (Windows/Linux/macOS) via Python/FastAPI backends. It is designed for millisecond-level to sub-second automation loops, sensor sampling, and instrument control. It is **not** intended for bare-metal FPGA hardware logic or hard real-time sub-microsecond control loops (e.g., fast motor drive PWM switching at 100 kHz). For such tasks, hardware FPGAs or dedicated microcontroller RTOS boards should handle low-level timing, while ComfyLAB communicates with them over serial/USB for high-level control, buffering, and plotting.

  Summary: Outside of microsecond hardware FPGA determinism, **virtually any scientific measurement, lab automation, data logging, signal analysis, or plotting workflow can be built in ComfyLAB**.

3. Comparison with Alternative Paradigms

Understanding how ComfyLAB compares to traditional lab software tools highlights why it is a compelling choice for both teaching and research.

Feature / Aspect	Pure Python	NI LabVIEW	MATLAB / Simulink	ComfyLAB
License & Cost	Free & Open-Source	Expensive commercial	Expensive commercial	**Free & Open-Source (GPLv3)**
Paradigm	Text-based script	Proprietary graphical G	Block diagram / script	**Web Block Diagram + Polyglot**
Learning Curve	High (syntax, APIs)	Steep (G conventions)	Moderate (toolboxes)	**Low (Visual drag-and-drop)**
GUI & Plotting	Manual (PyQt, Tkinter)	Built-in Front Panel	Built-in Figure windows	**Instant 2D/3D & Dashboard (`D`)**
Offline Simulation	Manual mock libraries	Basic software mocks	Simulink physics engine	**Built-in SCPI Virtual Instruments**
Files & Versioning	Plain text `.py` (Git-friendly)	Binary `.vi` (Merge conflicts)	`.m` / Binary `.slx`	**Clean JSON Blueprints (Git-friendly)**
Custom Code	Native Python	Complex C/Python nodes	S-Functions / MATLAB	**9-Language Script Blocks & DLL/SO**
Architecture	Monolithic script	Heavy desktop app	Heavy desktop suite	**Decoupled Web UI + FastAPI Server**
Hardware Locks	Manual implementation	Built-in DAQmx	Instrument Toolbox	**Automatic async VISA lock manager**

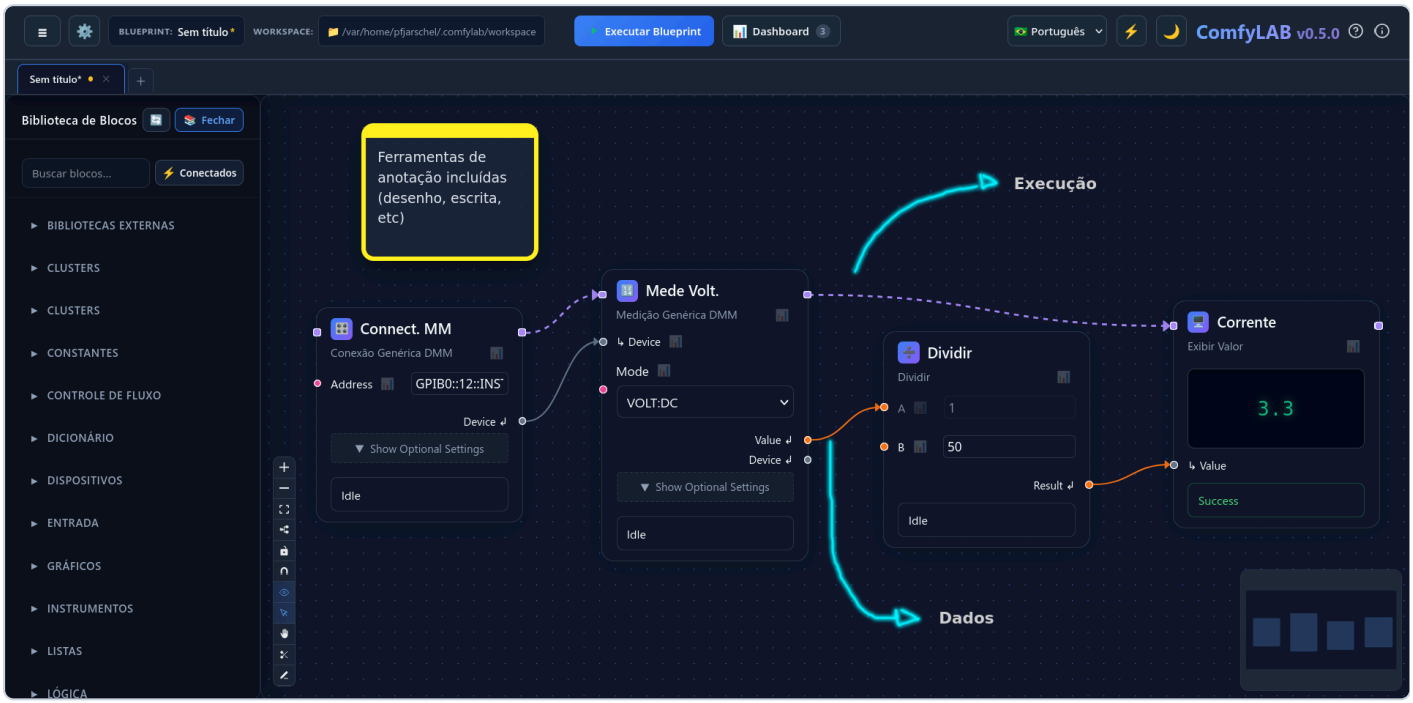
Key Takeaways:

1. **Vs. Pure Python**: Python is powerful, but creating user interfaces, real-time interactive plots, hardware state loops, and visual workflows requires hundreds of lines of boilerplate code. ComfyLAB keeps Python's full analytical power while providing an immediate visual UI and experiment dashboard.
2. **Vs. LabVIEW**: LabVIEW is widely used but suffers from expensive commercial licenses, heavy proprietary binary files that break version control, and vendor lock-in. ComfyLAB offers a modern, open-source alternative with JSON-based blueprints, built-in virtual instruments, and seamless web browser integration.
3. **Vs. MATLAB**: MATLAB excels at matrix computations but carries heavy licensing costs. ComfyLAB leverages free, open-source NumPy/SciPy backends under the hood.

4. Interface Layout & Core Concepts

4.1 Interface Layout Overview

The ComfyLAB user interface consists of three primary functional zones surrounded by top controls:



ComfyLAB Main Interface

Detailed Interface Description:

- **Top Toolbar**: Controls for running (▶️ / `Ctrl+R`), pausing (⏸️), and stopping (🛑 / `Ctrl+Shift+R`) blueprints, opening the **Experiment Dashboard** (📊 / `D`), creating/saving workspace files (`Ctrl+S`), changing color themes, and configuring global VISA settings and diagnostics.
- **Sidebar (Left)**: Searchable palette containing all available built-in blocks, custom published blocks, virtual instrument blocks, and script nodes categorized by domain.
- **Main Canvas (Center)**: Infinite zoomable and pannable visual workspace where you drag, position, and wire functional blocks together.
- **Block Inspector (Right)**: Contextual detail panel that opens when a block is selected. Displays live pin values, block parameters, brief documentation, notes, and execution console logs.
- **Tab Bar & Breadcrumbs**: Navigate between open blueprint files and drill down into sub-canvas clusters.

4.2 Pins & Wires: Execution Flow vs. Data Values

Connections between blocks in ComfyLAB are strictly divided into two distinct categories:

```
Exec Out  ▶═══════════════════════▶ Exec In
          (Violet Line: Controls WHEN blocks run)

Data Out  ●───────────────────────● Data In
          (Solid Line: Transfers VALUES on demand)
```

1. **Execution Pins (Flow)**:
    - Represented by **arrow-shaped handles** and connected via **violet lines (🟪)**.
    - Dictate **sequential timing and control flow** (when steps execute, loops repeat, or branches trigger).
2. **Data Pins (Values)**:
    - Represented by **circular colored pins** and connected via **solid colored lines**.
    - Carry **values** (numbers, text strings, NumPy vectors, dictionaries, hardware handles).
    - Evaluated lazily on demand when an execution block requests input data.

Data Pin Color Reference:

- 🟧 **Number (Orange)**: Scalar integer or floating-point numerical values.
- 🟩 **Boolean (Green)**: Logical `True` or `False` flags.
- 🟪 **String (Pink)**: Text strings, formatted characters, or SCPI commands.
- 🟨 **Array (Yellow)**: High-performance 1D or 2D NumPy numerical arrays/vectors.
- 🟦 **List (Cyan)**: Ordered collections of elements.
- 🟫 **Dictionary (Brown)**: Key-value parameter mappings.

4.3 Execution Logic & Independent Chains

- **Execution Trigger**: Clicking **Run Blueprint** (▶️ or `Ctrl+R`) starts execution at entry blocks (blocks with no incoming execution connections) and follows execution wires downstream.
- **Independent Execution Chains**: If the canvas contains separate, unconnected block groups, ComfyLAB automatically treats them as parallel execution chains, allowing independent acquisition loops to run concurrently.
- **Control Flow & Timing Blocks**:
    - **For Loop**: Executes a sub-loop for a specified iteration count, outputting current index, completion percentage (`%`), and estimated time remaining (`ETR`).
    - **For Each Loop**: Iterates through each item in a list or array, providing item, index, percentage, and ETR outputs.
    - **While Loop**: Continually executes a sub-loop while a Boolean condition pin remains `True`.
    - **If / Else Branch**: Routes execution flow to either a `True` or `False` output path based on a condition.
    - **Delay / Sleep**: Pauses execution flow for a specified duration (ms or seconds).
    - **Countdown Wait**: Pauses execution for a specified duration with an active live countdown clock, draining progress bar, and interactive **Skip Wait** button.
    - **Progress Bar**: Standalone visual display widget for tracking live completion percentage (0–100%).
    - **ETR Display**: Mission-control digital readout for estimating and rendering remaining run durations (`HH:MM:SS` or `MM:SS`).
    - **Execution Timer (Measure Time)**: Measures exact wall-clock elapsed time between trigger pulses.

4.4 Advanced Features

- **Experiment Dashboard & Pinned Cards (📊 / `D`)**: The Dashboard provides a clean, distraction-free "operator cockpit" for running experiments, presenting results, or conducting practical lab work without the visual complexity of block diagrams and wires.
    - **Pinning Blocks**: Any block or pin can be pinned to the Dashboard. Right-click a block and select **Pin to Dashboard**, click the pin icon in the Block Inspector header, or use the card pin icon on supported widgets.
    - **Automatic Organization**: Inputs, sliders, and constants are organized into interactive **Control Cards**, while graphs, gauges, indicators, and timers become dedicated **Monitor Cards**.
    - **Layout Customization**: Reorder cards up/down, drag corners to resize plots, toggle card controls, or click the jump icon to instantly navigate back to the corresponding canvas block.
    - **Maximized View**: Maximize the dashboard panel to fill the workspace for a streamlined full-screen lab control station.
- **Persistent Blocks (📌)**: By default, block states reset between execution runs. Marking a block as **Persistent** preserves its internal state and hardware handles (e.g., active VISA connections, open serial ports, or running averages) across multiple blueprint runs.
- **Disabling Blocks (🚫)**: Right-clicking a block to disable it dims the block on canvas. Disabled blocks are bypassed during execution, passing data through cleanly without running code—ideal for testing workflows without physical hardware.
- **Clusters (Sub-canvases 📦)**: Group complex block arrangements into a single custom block. Double-clicking a cluster block opens an isolated sub-canvas, with top breadcrumbs for navigation.
- **Whiteboard Mode (🎨)**: Click the tool icon or press hotkey `4` to toggle drawing mode, allowing freehand ink notes, text boxes, and shapes to be added directly over the canvas.

5. Summary of Built-in Block Categories

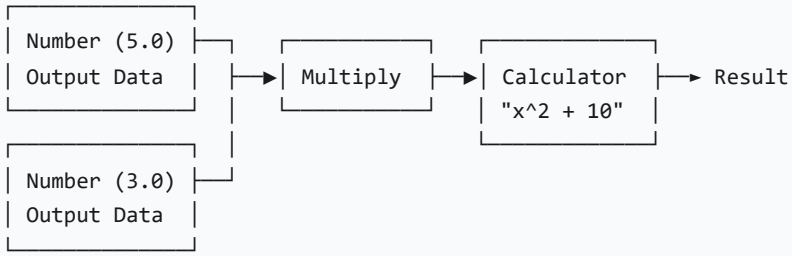
Category	Primary Function & Key Blocks
🔄 **Control Flow & Timing**	Loop orchestration and execution timing: `For Loop`, `For Each Loop`, `While Loop`, `If/Else`, `Sleep`, `Countdown Wait`, `Measure Time`, `Stop Execution`.
🔢 **Math & Logic**	Arithmetic, boolean logic, and formulas: `Add`, `Subtract`, `Multiply`, `Divide`, `Calculator`, `Trigonometry`, `Exponential`, `Log`, comparisons (`>`, `<`, `==`), `AND/OR/NOT`.
📝 **Data & Strings**	Text manipulation and structures: `String Concat`, `Format Text`, `Regex Search`, `String Split`, `List Builder`, `List Indexer`, `Dict Builder`, `Dict Get`.
📊 **Arrays & Signals**	Signal processing and arrays: `Create Array`, `Linspace`, `Array Slice`, `FFT Spectrum`, `Digital Filters` (low/high/bandpass), `Detrend`, `Peak Detector`, `Curve Fitting`.
💾 **File I/O**	Persistence and data export: `Save CSV`, `Load CSV`, `JSON I/O`, `Parquet Storage`, `Text File I/O`, `Image Loader/Saver`.
📡 **VISA & Physical Hardware**	Instrument communication: `VISA Device`, `VISA Query`, `VISA Read`, `VISA Write`, `Serial Port Open/Read/Write`, `Auto Resource Discovery`.
🖥️ **Virtual Instruments**	Zero-hardware simulation: `VirtOsc Connect/Acquire`, `VirtSigGen Connect/Config Wave`, `Simulated RC Circuit`.
🔬 **Device Drivers**	Vendor instrument drivers: Oscilloscopes (Tektronix, Keysight), Multimeters (HP/Agilent 34401A), Power Supplies, Horiba Spectrometers, CAEN Digitizers, Thorlabs Motors.
📈 **Display & Dashboard**	Data visualization and monitoring: `XY Plot`, `Dual Y-Axis Plot`, `Bar Plot`, `Box / Violin Plot`, `Histogram`, `3D Surface/Scatter`, `Polar Plot`, `Waterfall Spectrogram`, `Heatmap Plot`, `Progress Bar`, `ETR Clock`.
⚙️ **Scripting & Native**	Custom extensions: `Native Library Invocation` (C/C++ DLL/SO via Signature Editor), `Multi-Language Script Blocks` (Python, JS/TS, Julia, Rust, Lua, Octave, R, Wolfram).

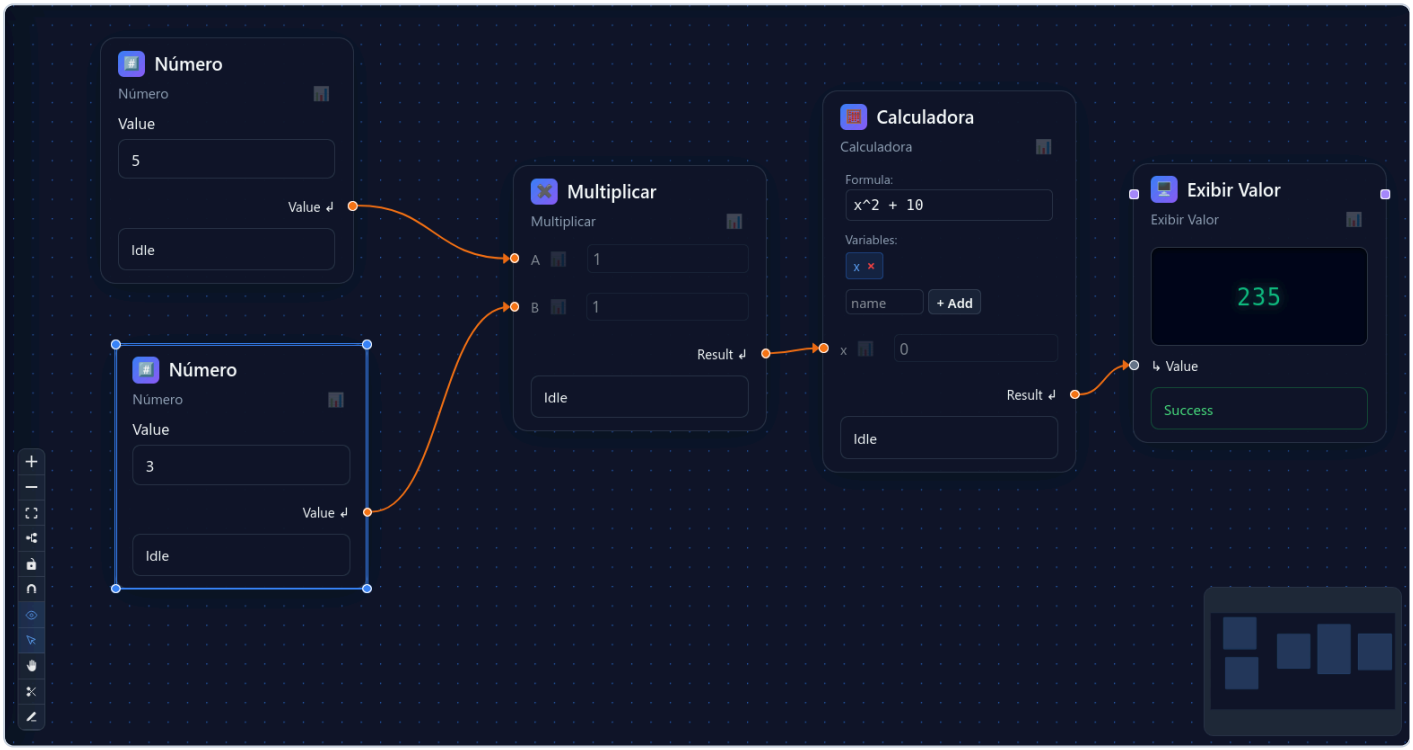
6. Hands-On Starter Tutorials

Follow these step-by-step practical examples to quickly master core ComfyLAB workflows and features.

Tutorial 1: Math Pipeline & Live Inspection

Goal: Create two numbers, multiply them, evaluate a mathematical expression using `Calculator`, and inspect the result in the Block Inspector.





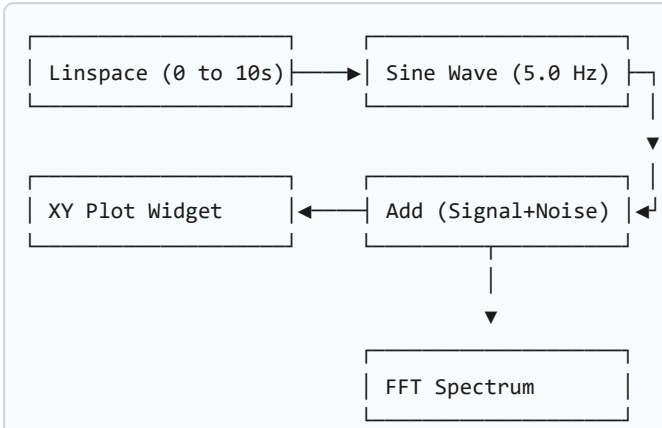
Tutorial 1 Workspace Example

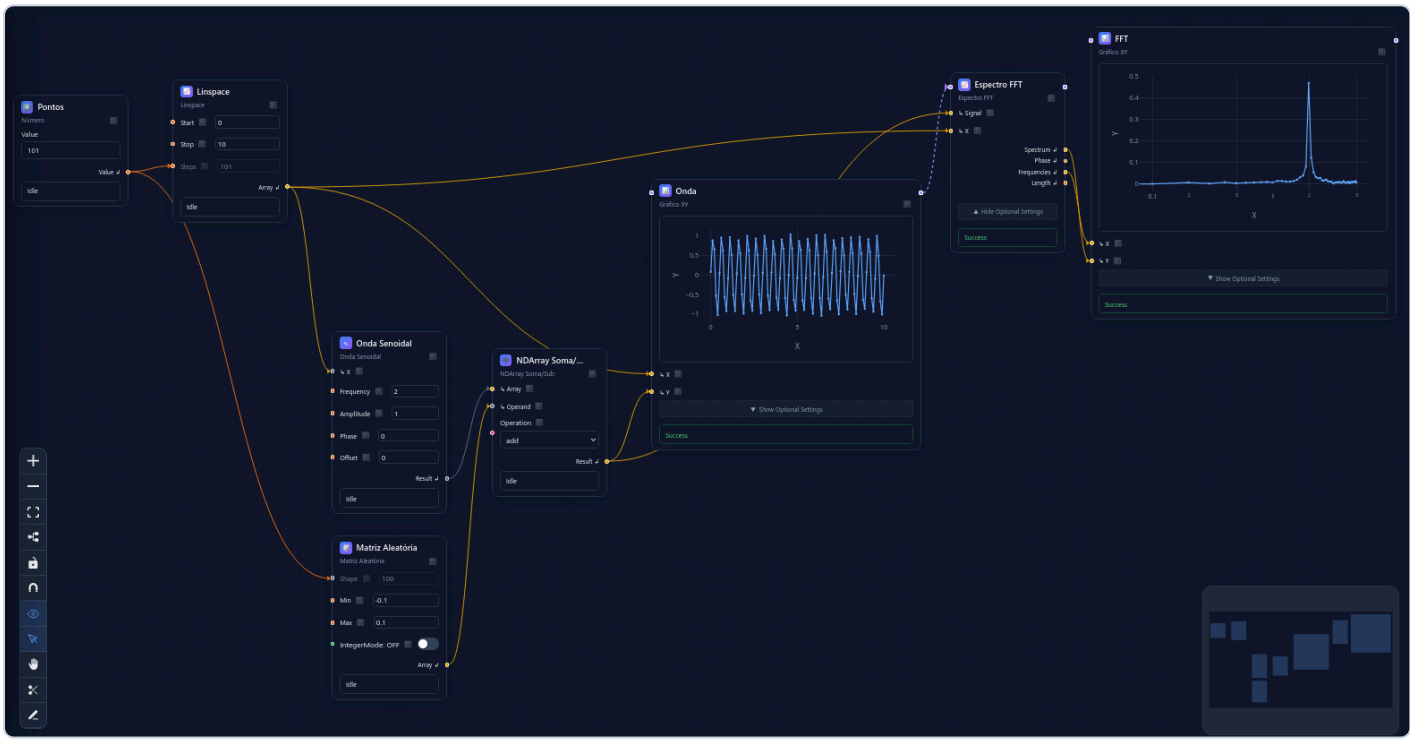
Step-by-Step Instructions:

1. Open ComfyLAB and create a new workspace/blueprint.
2. In the left **Sidebar**, search for `Number` (under *Constants*) and drag **two** Number blocks onto the canvas.
3. Set the value of the first Number block to `5.0` and the second to `3.0`.
4. Drag a **Multiply** block (from *Math / Basic*) onto the canvas.
5. Connect the data output pin of Number 1 to Input A of the Multiply block, and Number 2 to Input B.
6. Drag a **Calculator** block (from *Math / Basic*) onto the canvas. In the Block Inspector, set its expression to `x^2 + 10` and add variable input `x`.
7. Connect the output pin of the Multiply block (`15.0`) to input pin `x` of the Calculator block.
8. Click **Run Blueprint** (▶️ or `Ctrl+R`) on the top toolbar.
9. Click the **Calculator** block to open the **Block Inspector** on the right sidebar.
10. **Verification**: In the Block Inspector, check the live `Result` output pin value. It should evaluate to `(15)^2 + 10 = 235.0`.

Tutorial 2: Synthetic Signal Generation & Live Plotting

Goal: Generate a noisy sine wave array, calculate its FFT power spectrum, and render it on an `XY Plot`.





Tutorial 2 Workspace Example

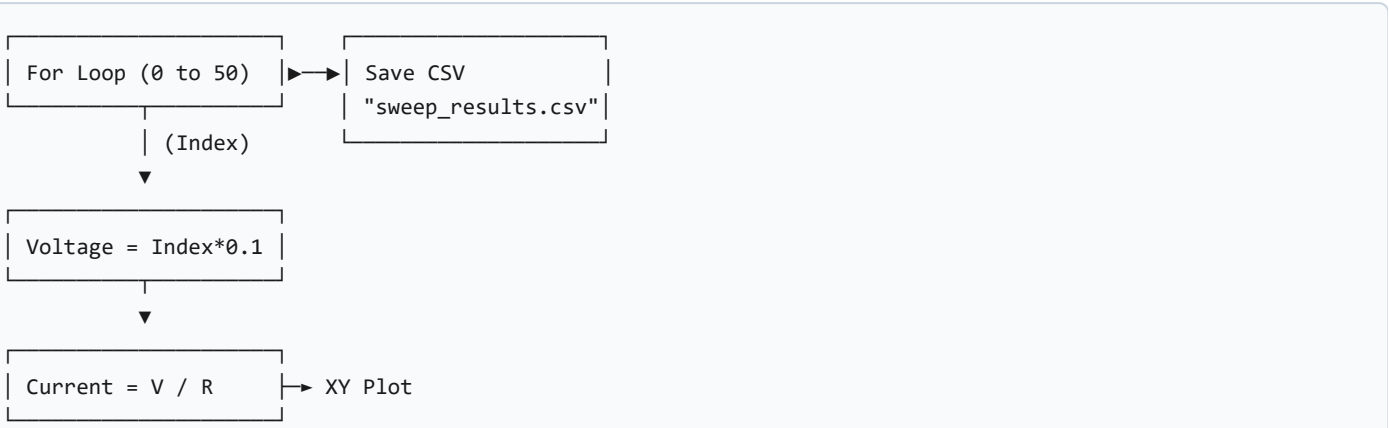
Step-by-Step Instructions:

1. Search for `Linspace` (under *Numeric Arrays*) in the Sidebar. Set `Start = 0`, `Stop = 10`, `Points = 1000` to create a time vector array.
2. Search for `Sine Wave` (under *Math / Functions*). Connect the Linspace output array to the `X` input of the Sine Wave block. Set `Frequency = 5.0` Hz.
3. Add a **Random Array** block (under *Math / Random*, set `Shape = 1000`, `Min = -0.2`, `Max = 0.2`) and an **Add** block to sum the sine wave with noise.
4. Drag an **XY Plot** block (under *Plots*) onto the canvas.
5. Connect the Linspace time array to the **X** pin of the XY Plot and the noisy signal array to the **Y** pin. Connect an execution wire to the `Plot` pin.
6. (Optional) Wire the signal to an **FFT Spectrum** block (under *Math / Signal Processing*) to view frequency peaks.
7. Click **Run Blueprint** (▶️ or `Ctrl+R`).

8. **Verification**: The XY Plot widget on your canvas will immediately display the live noisy sine wave. Use your mouse wheel over the graph to zoom in/out and drag to pan across axes!

Tutorial 3: Automated Parameter Sweep & CSV Logging

Goal: Use a `For Loop` to simulate sweeping a voltage, calculate current, and log data to a CSV file using `Save CSV`.



Step-by-Step Instructions:

1. Drag a **For Loop** block (under *Control Flow*) onto the canvas.
2. Set the `For Loop` iterations count to `50`. Notice that the block provides live `%` (percentage) and `ETR` output pins!
3. Inside the loop execution path, multiply the `Index` pin by `0.1` using a **Multiply** block to generate a voltage sweep from `0.0 V` to `5.0 V`.
4. Divide the voltage by a constant resistance (`R = 100 Ohms`) using a **Divide** block to calculate current `I`.
5. Drag a **Save CSV** block (under *File I/O*) onto the canvas. Set `FilePath` to `sweep_results.csv`.
6. Connect the data values to the `Data` pin of `Save CSV`. Connect the `Body` execution wire of the loop to the `Write` pin of `Save CSV`.
7. Wire the `Out` execution pin of `Save CSV` back to the loop `Step` input to advance the iteration.
8. Click **Run Blueprint** (▶️ or `Ctrl+R`).
9. **Verification**: Watch the loop run step-by-step with animated violet execution wires. After completion, check your active workspace folder—you will find `sweep_results.csv` populated with all 50 data rows!

Tutorial 4: Writing & Publishing a Custom Python Script Block

Goal: Create a custom Python script block that filters out values below a threshold, test it, and publish it to the left sidebar palette.

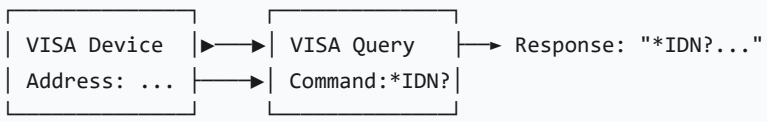
```
┌──────────────────────────────────────────────┐
│ Custom Script Block (Python)                 │
│ Inputs:  data_in (Array), threshold (Number) │
│ Outputs: filtered_data (Array), count (Number)│
├──────────────────────────────────────────────┤
│ arr = np.array(inputs['data_in'])            │
│ result = arr[arr > inputs['threshold']]      │
│ outputs['filtered_data'] = result.tolist()   │
│ outputs['count'] = len(result)               │
└──────────────────────────────────────────────┘
```

Step-by-Step Instructions:

1. Drag a **Python Script** block (from *Scripts*) onto the canvas.
2. In the **Block Inspector**, click **Edit Pins**:
    - Add Input Pin: `data_in` (Type: Array)
    - Add Input Pin: `threshold` (Type: Number)
    - Add Output Pin: `filtered_data` (Type: Array)
    - Add Output Pin: `count_passed` (Type: Number)
3. Double-click the Python Script block to open the built-in Monaco code editor.
4. Enter the Python code snippet shown above. (Notice that `os.environ["COMFYLAB_WORKSPACE"]` is also available if your script needs local files!).
5. Connect a sample array to `data_in` and set `threshold = 2.5`.
6. Click **Run Blueprint** (▶️) and inspect `filtered_data` and `count_passed` in the Block Inspector.
7. Once verified, click **Publish Block** in the Block Inspector. Name it `Threshold Filter`, choose an icon, and assign it to the `Arrays` category.
8. **Verification**: Look at your left **Sidebar** palette. Your newly created `Threshold Filter` block is now permanently available to drag onto any canvas in your workspace!

Tutorial 5: VISA Hardware Interfacing (SCPI Identification Query)

Goal: Connect to a physical or simulated instrument over VISA using `VISA Device`, send an SCPI `*IDN?` query using `VISA Query`, and display the instrument model response.

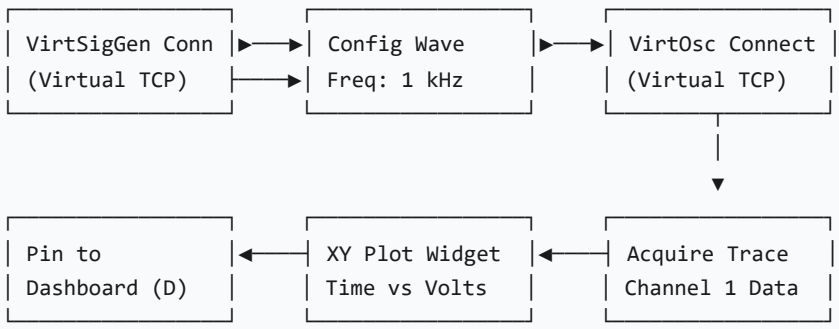


Step-by-Step Instructions:

1. Drag a **VISA Device** block (under *VISA / Core*) onto the canvas. Set the `Address` parameter to your instrument address (e.g. `TCPIP0::192.168.1.100::INSTR`, `GPIB0::14::INSTR`, `COM3`, or `VIRT::OSC`).
2. Drag a **VISA Query** block (under *VISA / Core*) onto the canvas.
3. Connect the `Out` execution pin of VISA Device to the `In` execution pin of VISA Query.
4. Connect the `Device` handle output pin (cyan pin) from VISA Device to the `Device` input pin of VISA Query.
5. Set the `Command` parameter in VISA Query to `*IDN?`.
6. Click **Run Blueprint** (▶️ or `Ctrl+R`). (ComfyLAB automatically manages connection caching, locks, and safe teardown).
7. **Verification**: Click the **VISA Query** block and check the Block Inspector. The `Response` output pin displays the instrument identification string (e.g. `HEWLETT-PACKARD,34401A,0,11-5-2` or `KEYSIGHT TECHNOLOGIES,DSOX2002A,...`).
8. **Tip**: You can also use the **VISA Resource Manager** block to discover connected physical instruments automatically!

Tutorial 6: Offline Automation with Built-in Virtual Instruments

Goal: Connect to ComfyLAB's embedded virtual instruments, configure a simulated signal generator, capture a waveform with the virtual oscilloscope, and monitor the results on the Dashboard without any physical hardware.



Step-by-Step Instructions:

1. From the Sidebar, navigate to **Devices > Virtual**.
2. Drag a **VirtSigGen Connect** block and a **VirtSigGen Config Wave** block. Connect them to generate a 1.0 kHz sine wave at 2.0 Vpp.
3. Drag a **VirtOsc Connect** block and a **VirtOsc Acquire** block. Connect the execution flow and the device handles.
4. Drag an **XY Plot** block. Connect the oscilloscope's time array output to **X** and the voltage trace array to **Y**.
5. Right-click the **XY Plot** block and select **Pin to Dashboard**.
6. Press `D` on your keyboard (or click **Dashboard** in the top toolbar) to open the Operator Dashboard panel.
7. Click **Run Blueprint** (▶️ or `Ctrl+R`).
8. **Verification**: ComfyLAB will automatically spawn the background virtual instrument simulation server. The Operator Dashboard will immediately display the live captured waveform. You can also load the pre-built **Bode Diagram Virtual** workflow from **File Menu > Load Example > Bode_Diagram_Virtual.json** to see a full automated frequency sweep with Bode plots!

7. Cheat Sheet & Keyboard Reference

Hotkeys & Shortcuts

Action	Shortcut	Description
Toggle Dashboard	`D`	Open or close the operator Dashboard cockpit
Run / Pause / Resume	`Ctrl + R`	Start blueprint execution, pause, or resume
Stop Execution	`Ctrl + Shift + R`	Abort execution immediately and run safety shutdown hooks
Duplicate Block(s)	`Ctrl + D`	Clone selected blocks preserving configured parameters
Select Tool	`1`	Default mode: select, move blocks, and connect wire pins
Pan Tool	`2`	Pan canvas viewport without selecting blocks
Cut Wire Tool	`3`	Slice across wires with cursor blade to delete connections
Whiteboard Tool	`4`	Toggle drawing overlay for notes, shapes, and freehand ink
Pan Canvas	`Space + Drag`	Hold spacebar and drag background to pan
Zoom Canvas	`Mouse Wheel`	Scroll wheel to zoom in or out centered at cursor
Save Blueprint	`Ctrl + S`	Save current blueprint (`Ctrl + Shift + S` for Save As)
Open Blueprint	`Ctrl + O`	Open file dialog to load blueprint JSON
Undo / Redo	`Ctrl + Z` / `Ctrl + Y`	Undo or redo recent canvas actions
Copy / Paste	`Ctrl + C` / `Ctrl + V`	Copy and paste selected blocks
Delete	`Delete` / `Backspace`	Remove selected blocks or wires
Enter Cluster	`Double Click`	Drill down into Cluster sub-canvas