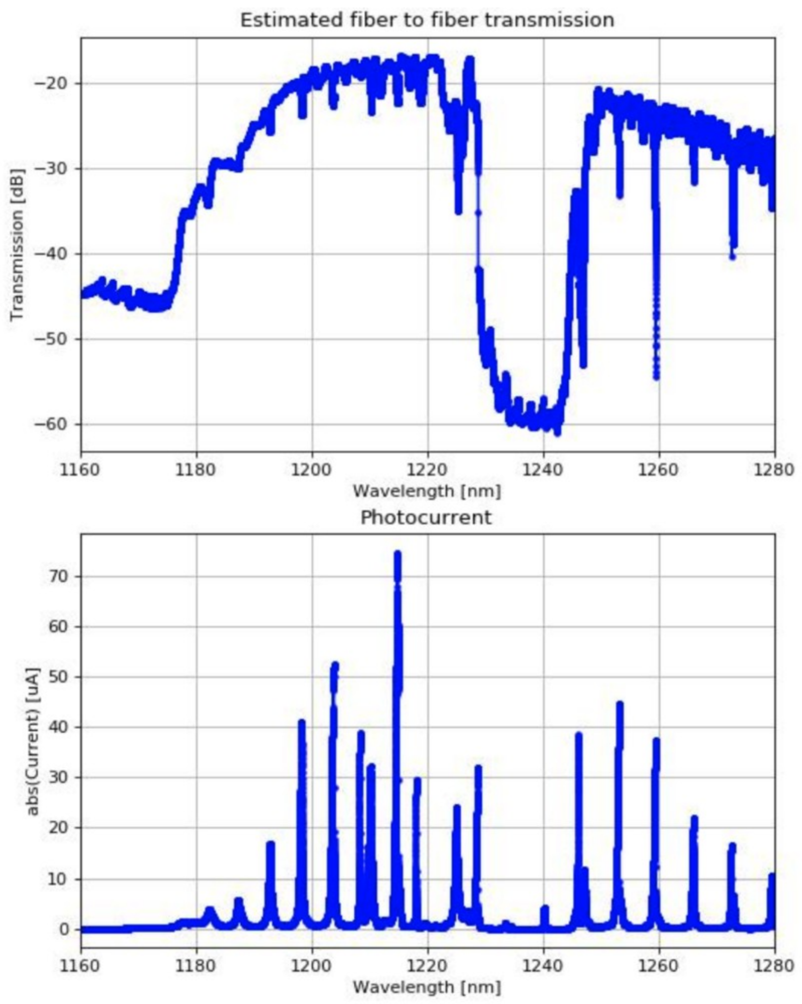
# Laboratory test and measurement automation

Luca Alloatti, luca.alloatti@ief.ee.ethz.ch

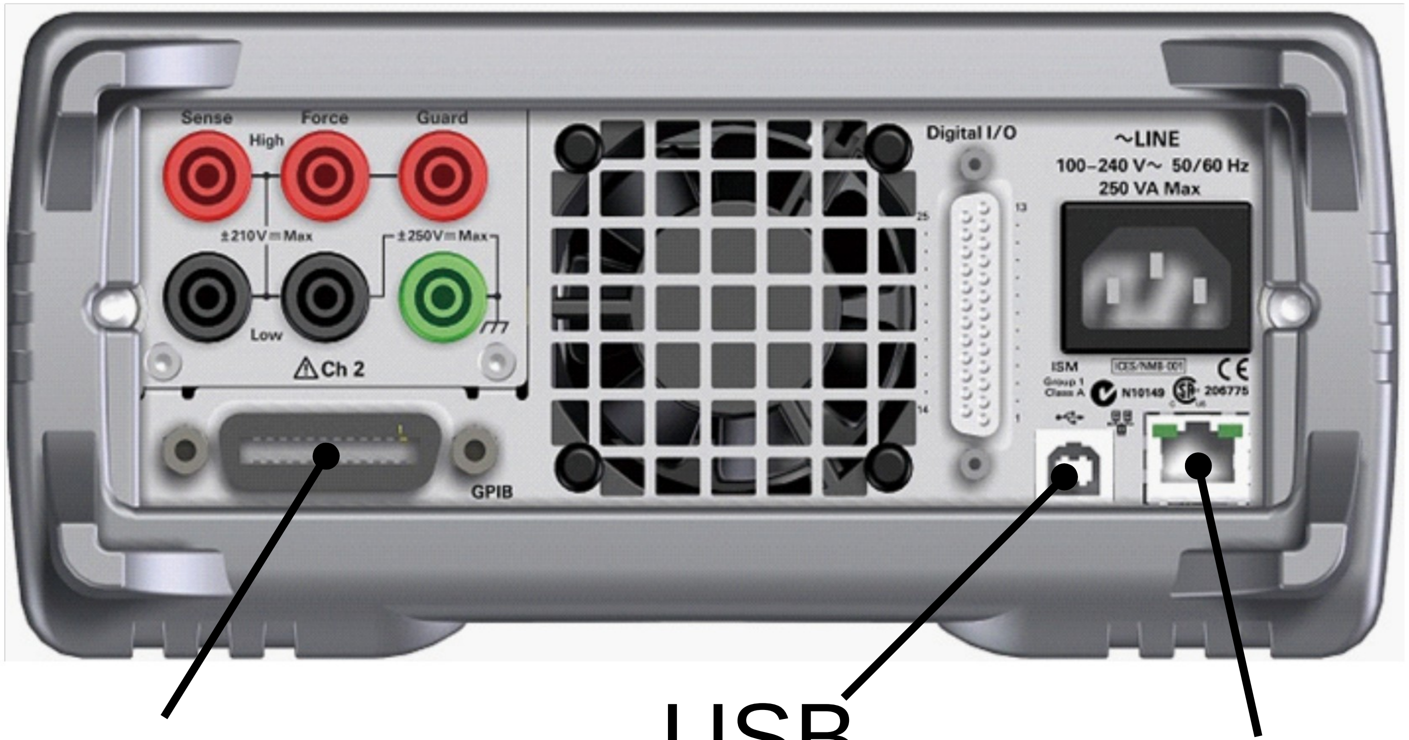
ETH zürich

## A typical lab experiment

- A typical experiment requires several machines to work in concert.
- On the right is the example of a typical experiment: the graphs show the optical spectrum and the photocurrent of a resonant photodiode. For this experiment a laser, two optical powermeters and and an electrical powermeter had to work together.
- Other experiments in our laboratory, for example, measure hundreds of devices without assistance, requiring automatic alignments with sub-micrometre accuracies: microstages and feedbacks work together under the supervision of a computer.



## Typical interfaces of laboratory equipment



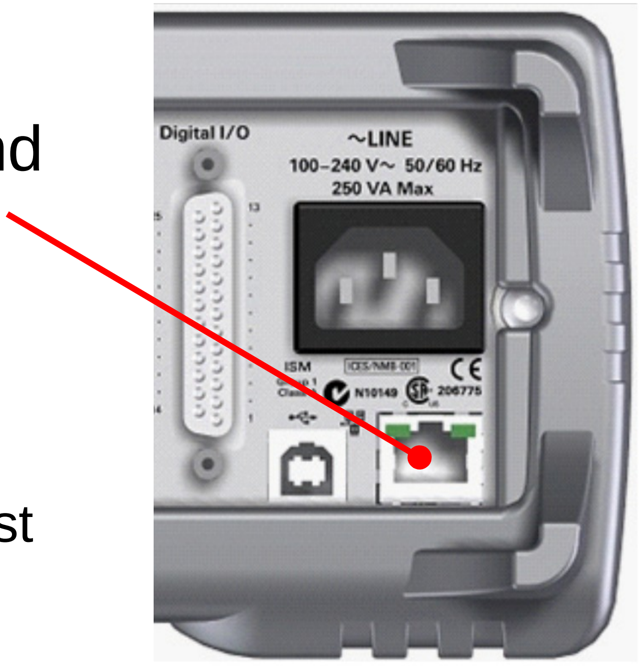
**GPIB**
Historical, still common on most devices

**USB**
More rare, not suited in a lab with several shared instruments

**Ethernet**
is becoming the new standard

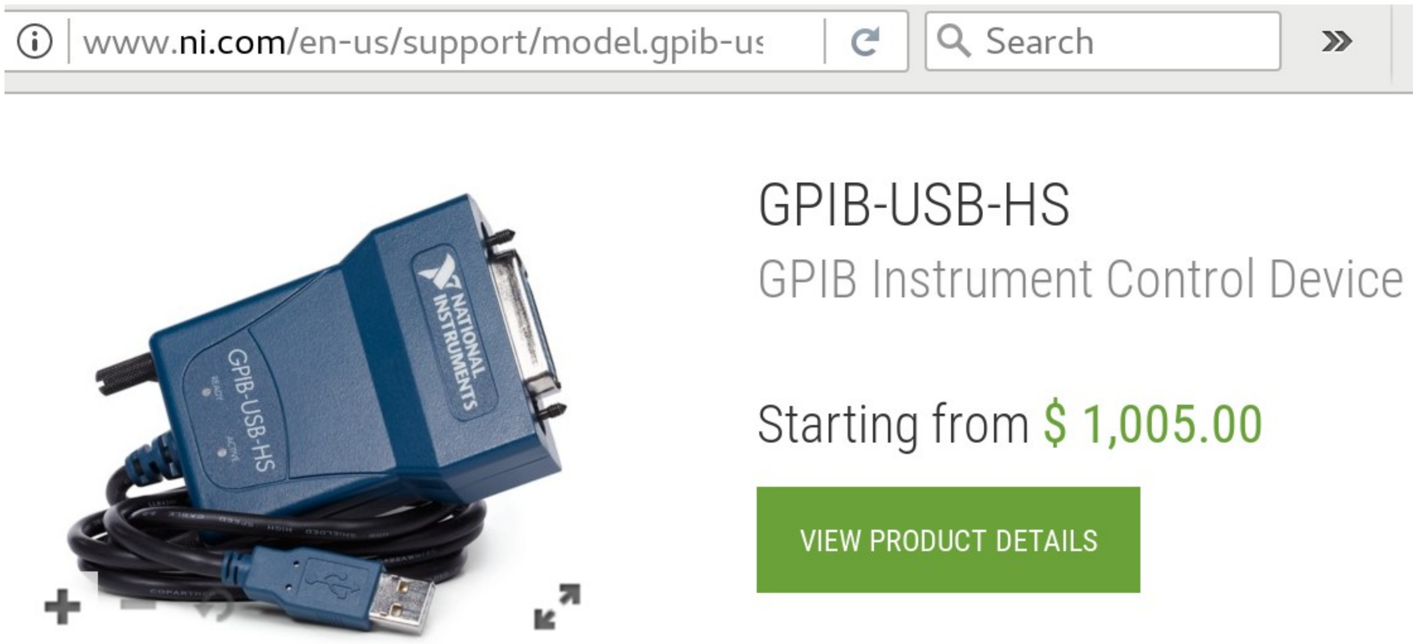
## LAN eXtensions for Instrumentation (LXI)

- LXI is a **standard** which defines the **communication protocols** for test and automation equipment using Ethernet
- It is defined and promoted by the “LXI consortium” which is a **US non-profit organization**.
  - The LXI standard is relatively young: the first LXI meeting was in 2004
  - Key members: Keysight Technologies, Pickering Interfaces and Rohde & Schwarz



## National Instruments (NI) GPIB-USB interface

- Over the past two decades several laboratories utilized the GPIB-USB “High Speed” converter to connect a GPIB connector of an instrument with the USB port of a computer.
  - This adapter is however very expensive
  - It requires proprietary drivers on the PC
  - It encourages use of LabView



GPIB-USB-HS

GPIB Instrument Control Device

Starting from $ 1,005.00

ETH zürich

## Since a few years (2010+ ?):

- Since a few years a new GPIB adapter of the company Prologix appeared on the market (image)
  - This device is significantly cheaper than the NI one (200$ vs. 1000$)
  - It supports the LXI Ethernet interface
  - It is simple to use in combination with open-source tools
  - It enables to “convert” old instruments with **GPIB interface only**, into LXI Ethernet instruments

ETH zürich

## The General Purpose Interface Bus (GPIB interface)

- The GPIB is a short-range, digital, multi-master, interface bus specification
- It is an 8-bit, electrically parallel bus. The bus employs **sixteen** signal lines: eight used for bi-directional data transfer(three for handshake, and five for data) plus eight ground return lines
- Its connector allows to stack several connectors on each others (stacking connectors). Very practical in a constantly-changing laboratory

[1] History of GPIB, http://www.ni.com/white-paper/3419/en/

ETH zürich

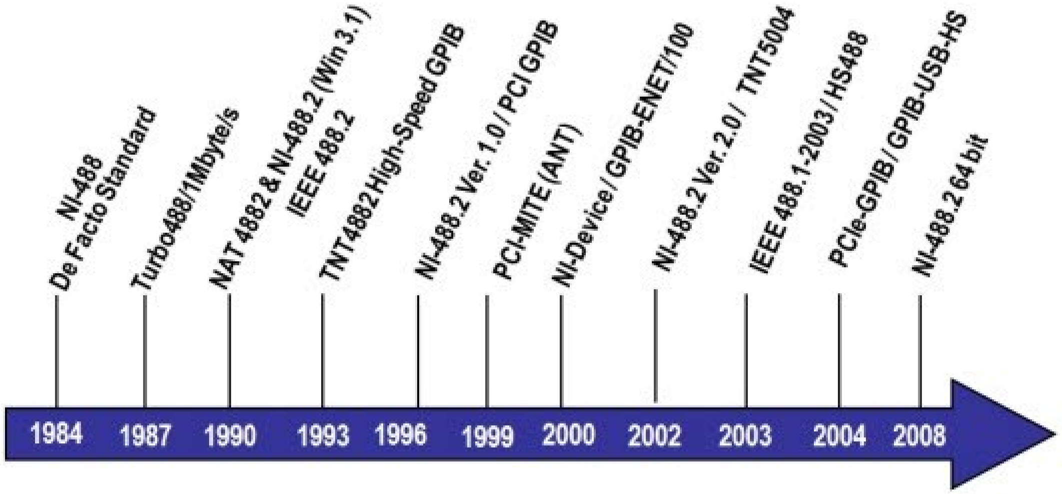
## Some history of the GPIB

- Late 1960s: Hewlett Packard (HP) develops the original GPIB interface for its instruments back then called HP-IB [1]
- 1975: the Institute of Electrical and Electronic Engineers (IEEE) published ANSI/IEEE Standard 488-1975, IEEE Standard Digital Interface for Programmable Instrumentation, which contained the electrical, mechanical, and functional specifications of an interfacing system. The original IEEE 488-1975 was revised in 1978, primarily for editorial clarification and addendum.
- This bus is now used worldwide and is known by three names: HB-IB, GPIB and IEEE 488 Bus

[1] History of GPIB, http://www.ni.com/white-paper/3419/en/

## Some history of the GPIB: the role of National Instruments

- 1976: National Instruments is founded [1]
- NI has since 1976 been devoted to use GPIB
- NI produces software solutions, in particular LabView, which is still the standard in many research laboratories
- In NI's own words: << Today, NI has established itself as a leader in GPIB interfaces and Instrument Control software.>> [1]

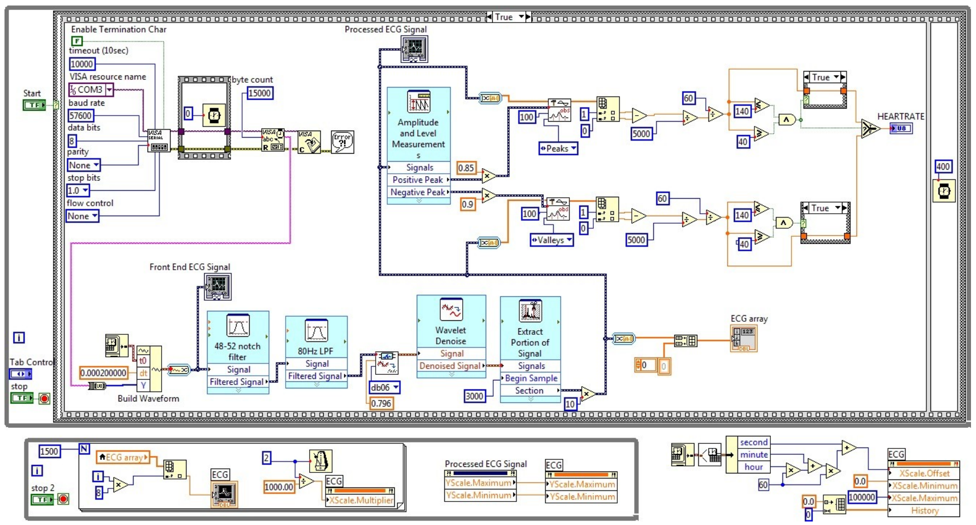


[1] History of GPIB, http://www.ni.com/white-paper/3419/en/

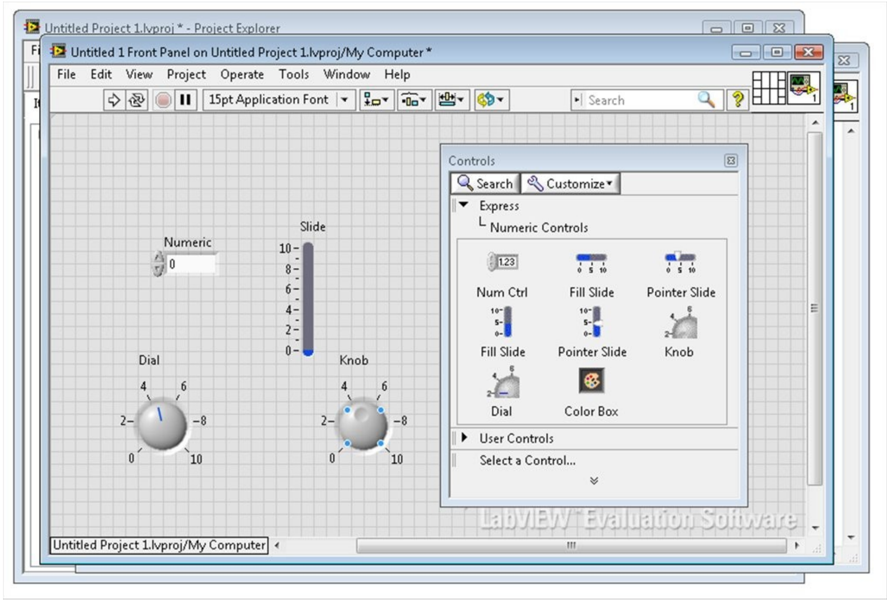
# NI's LabView

- LabView is possibly still the most used lab automation software
- It has a unique programming interface based on icons, and wires which symbolize buses

Programming interface:



User interface:


Untitled Project 1.lvproj * - Project Explorer
Untitled 1 Front Panel on Untitled Project 1.lvproj/My Computer *
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
Numeric
Slide
Dial
Knob
Controls
Search
Customize
Express
Numeric Controls
Num Ctrl
Fill Slide
Pointer Slide
Knob
Dial
Color Box
User Controls
Select a Control...
LabVIEW Evaluation Software
Untitled Project 1.lvproj/My Computer

## Alternatives to LabView

- Open-source alternatives based on Python are becoming increasingly popular because:
  - They are cheaper, more flexible, lighter
  - They do not require learning one more language (like LabView)

- Even large conferences started organizing workshops on lab automation based on Python:
  - https://python4photonics.gitlab.io/hackathon/
  - https://python4photonics.gitlab.io/blog/ofc-2018-labautomation-hackathon/

- Some groups use Matlab instead:
  - Matlab is scripted as Python and has a good support
  - However it is expensive

## Standard Commands for Programmable Instruments (SCPI commands)

- The word “SCPI” is often pronounced “skippy”
- The SCPI commands are basically universal across all laboratory equipment. Even though each instrument can define its own dialect to implement its unique features, the syntax is unique and it is defined by a standard.
- The SCPI standard is first released in 1990 as an additional layer on top of IEEE:
  - IEEE-488.1 specified the physical and electrical bus, and IEEE-488.2 specified protocol and data format, but neither specified instrument commands
- The SCPI standard builds on top of the “TML language” which was developed by HP in 1989
- Examples of SCPI commands:

  **“:SOUR2:FUNC:MODE CURR”** (set channel 2 of the electrical source into current mode)
  **“*RST”** (reset the instrument)

  **“*IDN?”** (ask the identity of the instrument)

## SCPI syntax

- According to the SCPI standards, all commands belong to the following two categories:
  - **set (or write):** used to perform a “set” operation. Example: “:SOUR1:VOLT3” (sets on a voltage source on channel 1 the voltage of 3 V)
  - **queries:** used when an answer from an instrument is expected. Query commands end with a “?” Examples:
    - “*IDN?” (ask the identity of the instrument)
    - “:SENS:DATA?”  (ask to read the data)

## SCPI syntax

- Commands have a short and a full form which can be used interchangeably,
  - for example:
    “:SOUR1:VOLT3”  is equivalent to:
    “:SOURce1:VOLTage3”
  - The part which is not an abbreviation is usually written lower case.
  - Abbreviations have often four characters, but NOT always

## SCPI syntax

- Commands are organized into a **hierarchy** and different parts of the command (not to be confused with the arguments) are separated by a colon ":"
  - The first colon indicates the "**root**" of the command tree (this is similar to the "/" in the linux)
    - Example: ":SYST:TIME:TIM:COUN:RES" (resets timer)

## SCPI syntax

- Some commands require some additional arguments, such as numbers or some text. Arguments are separated by a **space**
  - Examples:
    “:SOUR:FUNC:MODE  VOLT”
    “:SOURce:FREQuency:STOP 200”

## SCPI syntax

- If a the first colon is not given (root), the command is rooted in the **last node** of the previous command:
  - Example:
  - “:SURce:FREQuency:STARt 100”
  - “STOP 200”  (interpreted as “:SOURce:FREQuency:STOP 200”)

## SCPI syntax

- Commands starting with an asterisk “*” are special and are never preceded by a colon
  - Examples:
    - “*IDN?”
    - “*RST”

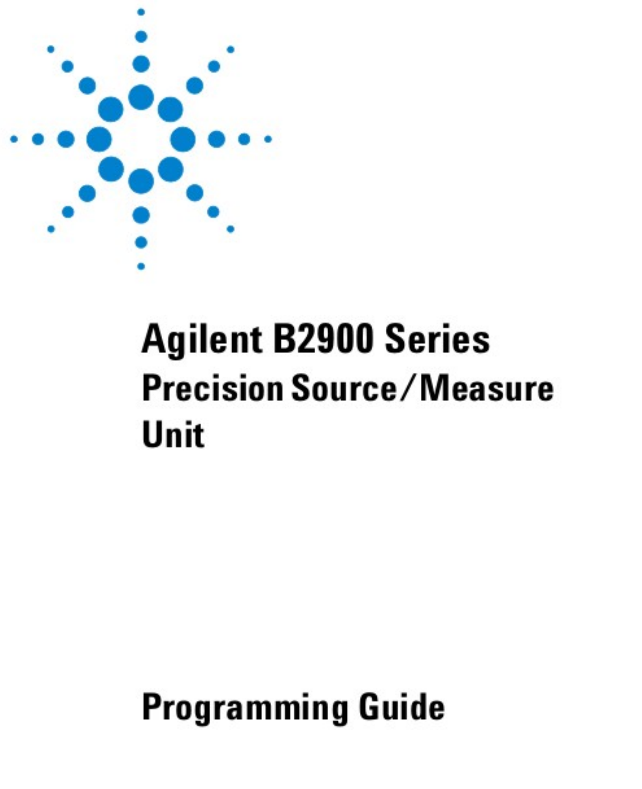
ETH zürich

## SCPI syntax

- Multiple commands can be **concatenated** into a **single** string by separating them with a semicolon “;”
  - For example: “:SOUR1:VOLT 3;*IDN?”

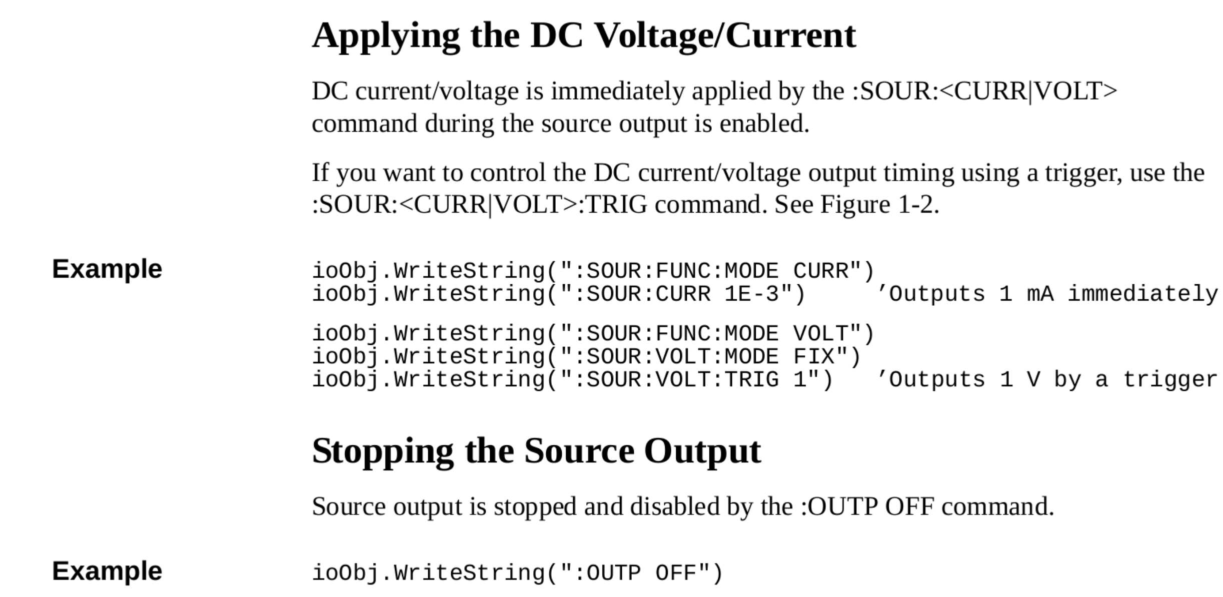
## Each instrument has its own SCPI commands

- Each instrument comes with a “programming manual” with all the details and examples.



Agilent B2900 Series
Precision Source/Measure
Unit

Programming Guide



### Applying the DC Voltage/Current

DC current/voltage is immediately applied by the :SOUR:<CURR|VOLT> command during the source output is enabled.

If you want to control the DC current/voltage output timing using a trigger, use the :SOUR:<CURR|VOLT>:TRIG command. See Figure 1-2.

**Example**

```
ioObj.WriteString(":SOUR:FUNC:MODE CURR")
ioObj.WriteString(":SOUR:CURR 1E-3")     ’Outputs 1 mA immediately

ioObj.WriteString(":SOUR:FUNC:MODE VOLT")
ioObj.WriteString(":SOUR:VOLT:MODE FIX")
ioObj.WriteString(":SOUR:VOLT:TRIG 1")   ’Outputs 1 V by a trigger
```

### Stopping the Source Output

Source output is stopped and disabled by the :OUTP OFF command.

**Example**

```
ioObj.WriteString(":OUTP OFF")
```

**Now solve the exercises in the Jupyter-notebook Exercise_11.ipynb**