University of Applied Sciences - Fribourg

Electrical and Electronics Engineering Faculty

Semiconductor devices

Installation of a development environment for Python

Author : Armando Bourgknecht

Last update : March 1, 2022

Table of Contents

1 Introduction

This document serves as a quick start reference to install and learn to use a full Python environment.

The report mainly covers the installation steps for computer running *Windows* operating system, however, steps can usually be reproduced for *Mac OS* and *GNU/Linux* computers without encountering significant difficulties.

At the end of this document, the user will have an *Anaconda* distribution to manage virtual environments, a Python interpreter, a Pyhon integrated development environment (IDE) for scientific computing (Spyder) and a version of *NI-VISA*, a standard instrumentation backend to configure, program and troubleshoot various devices like oscilloscopes.

2 NI-VISA installation

Due to the software's considerable size, the first step of this tutorial is to download and install the *NI-VISA* standard instrumentation backend.

The installer can be found here `https://www.ni.com/en-us/support/downloads/drivers/download.ni-visa.html#442805`.

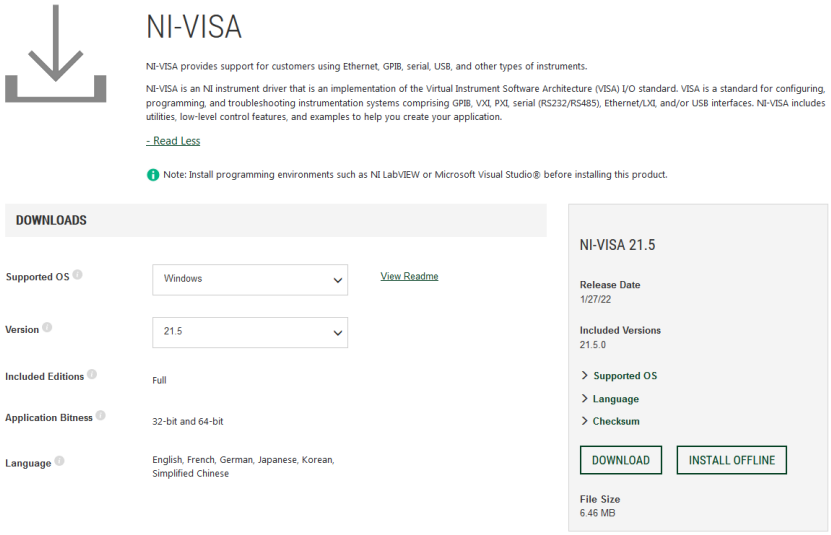

Figure 1: Downloading the NI-VISA installer.

3 Anaconda environment installation

This tutorial covers installation for the *Anaconda* distribution.

Anaconda is a distribution for the Python programming language for scientific computing, that aims to simplify package management and deployment.

3.1 Where to get the installer

Visit `https://www.anaconda.com/products/individual` to download the installer for the *Anaconda Individual Edition*. Multiple operating systems are supported:

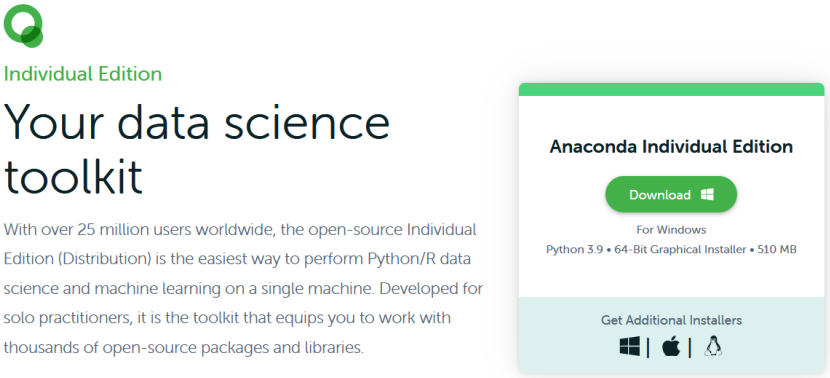

Figure 2: Download the Anaconda Individual Edition installer.

Launch the installer to go through software installation.

3.2 Managing virtual environments in Anaconda

Developing in Python often implies using different libraries for specific tasks, such as numerical computations, electronic instruments interfacing, machine learning and many others.

Each project being different, there might be a requirement to develop a program with the same library, but on a different version, implementing the newest and cutting edge software, or on an older and stable one.

To juggle between each programming context, Python implements *virtual environment* to create independent stand-alone environments :

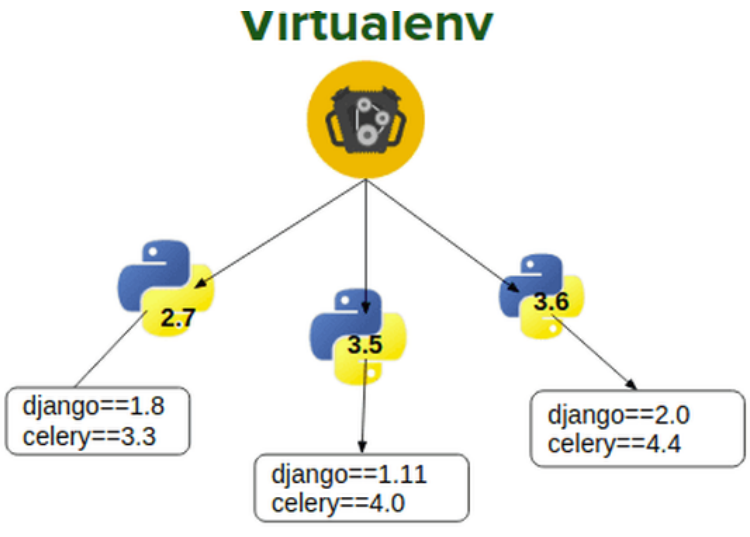

Figure 3: Python environments for different version and with different libraries.

The previous example shows multiple virtual environments to develop projects using older version of Python and the `django` and `celery` web development libraries.

Managing environments can either be done through the command line, but to simplify the process, we will use the graphical interface proposed by Anaconda Navigator.

3.2.1 Creating a virtual environment

On the Anaconda Navigator home page, locate the *Environments* tab on the left and click on it:

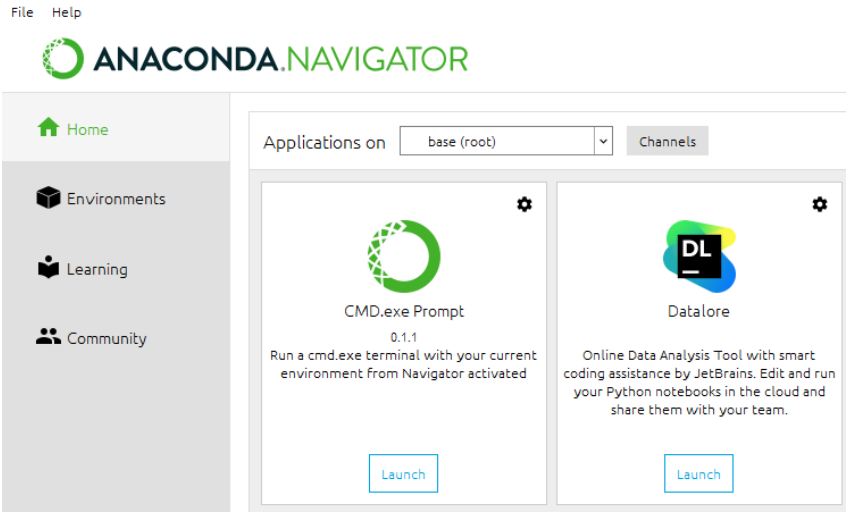

Figure 4: Anaconda Navigator home page with the *Environments* tab.

Then, locate the *Create* button on the bottom and click on it to add an environment:

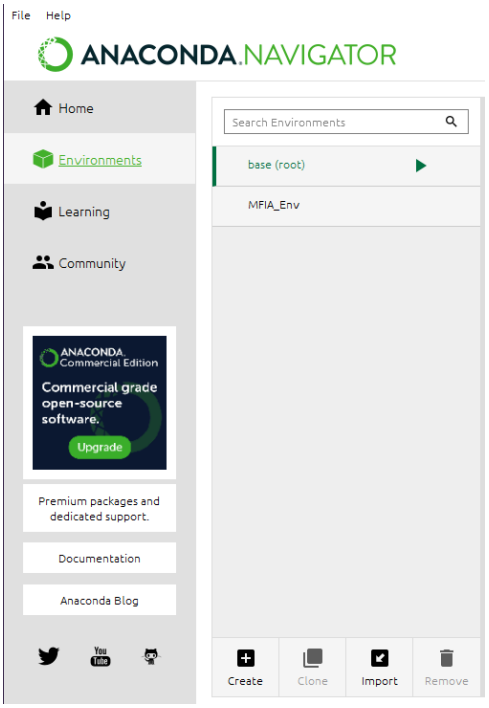

Figure 5: Environments management panel.

On the following window, enter the new environment name and select the appropriate Python version :

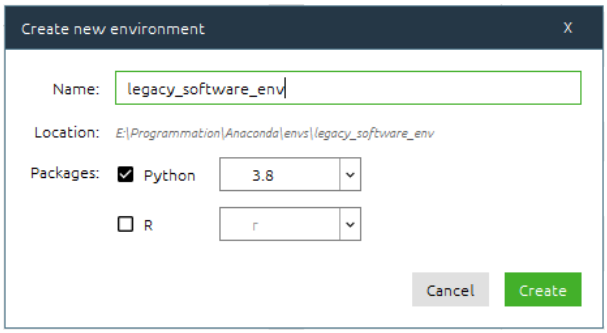

Figure 6: Environment creation with version selection.

In this example, I already have created some environments for other projects. For this course, we can create a single environment for all practical labs, like `TP_DAS`, where the useful software will be in common.

It is a good idea to get a clear view of the project's software requirements prior to creating the environment to guarantee compatibility between the different Python versions and libraries.

The environment is then created and can be chosen from the drop menu in the *Home* tab:

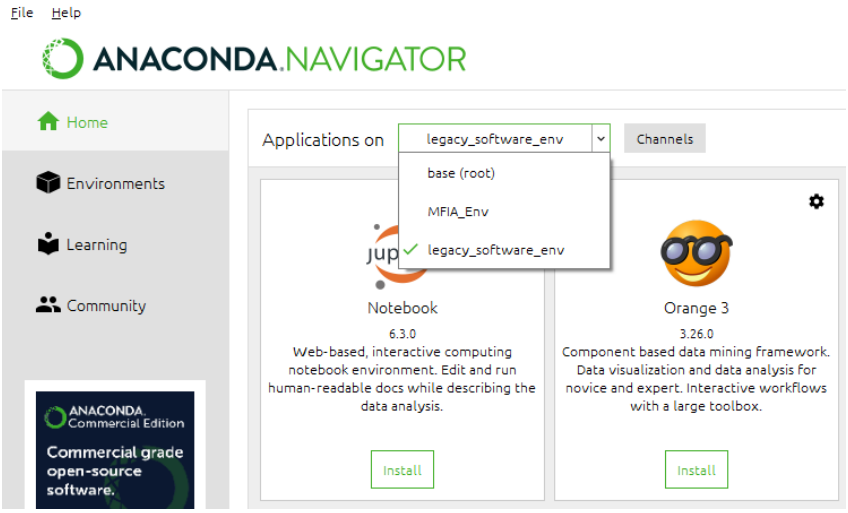

Figure 7: Selecting the newly created virtual environment.

It is worth noting that each of the programs present on the home tab is environment specific to further guarantee version management, so it will be required to install the *Spyder* IDE for each virtual environment.

4 Spyder installation

Spyder is a Python IDE including a live Python console, text editor with syntax highlighting, variable explorer, integrated plotter, debugger and many more features. Although it might seem superfluous to run a simple script, it is ideal to start programming quickly without spending time setting up a complex development environment and provides an entry point for future Python developers in order to shorten the familiarization process.

One the Anaconda Navigator page, locate the Spyder package and click on the install button to launch installation.

4.1 Running a Python program with Spyder

Once the installation is finished, launch Spyder.

Open the target script in the Spyder IDE and simply click on the run button to run the whole script:

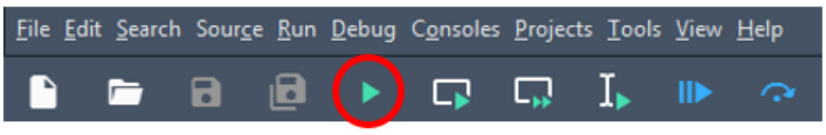

Figure 8: Running the whole script.

A highly useful feature of Spyder is the possibility to only execute parts of the script,

called *cells*. A *cell* is a section of code delimited by a line of comment beginning with two percentage symbols and ending at the next one:

```
# %% Imports

import pyvisa as visa # interface pour instrumentation

# %% Paramètres

# Connecter le câble USB
# Trouver l'adresse USB du MSOX3034T
# Bouton "Utility" > onglet "E-S" / "I/O"
mso_address = "USB0::10893::6000::MY55440267::INSTR"

# %% Connexion et initialisation

rm = visa.ResourceManager()         # Object de gestion des équipements
mso=rm.open_resource(mso_address)   # Connexion à l'appareil

mso.write("*IDN?")      # Demande l'identificatif de l'appareil
print(mso.read())       # Affichage de la reponse

mso.write("*CLS")       # Initialisation
mso.write("*RST")       # Remise à zéro de la configuration
```

Figure 9: Example of two cells delimited with a comment and two percentage symbols.

Individual cells can be run by clicking on the *Run Cell* button or with the useful `Ctrl-Enter` keyboard short cut :

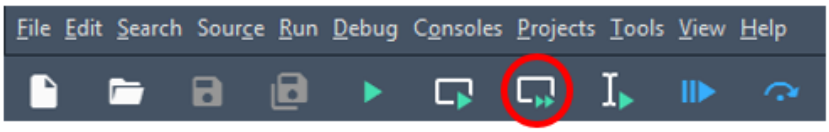

Figure 10: Running the selected cell only.

It is a rather convenient practice to segment the code into individual cells to test code more efficiently or avoid running a whole lengthy script from the beginning.

4.2 Using pip to install libraries

A common error when using Python is to try and import a not yet installed library in a project:

```
In [2]: import numpy as np
```

Figure 11: Import the *numpy* library without having it installed before hand.

The console will throw an error because it cannot find the library:

```
ModuleNotFoundError: No module named 'numpy'
```

Figure 12: Module not found.

This can be remedied by downloading the required library through the library management program `pip`, which is installed by default with Anaconda Navigator:

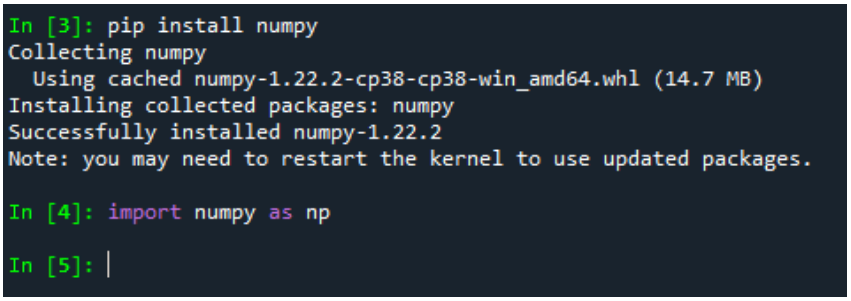

Figure 13: Installing missing dependency using pip and successfully importing it.

Usually, developers will package their libraries using `pip` to allow easy installation and removal, and installing a library without using `pip` is not advised except for experimented developers.

For example, NI-VISA has packaged a library for Python which can be found here : `https://pypi.org/project/PyVISA/`.

Once the NI-VISA standard backend is installed, the `pyvisa` library can simply installed by typing the following:

Installation of pyvisa

```
1 pip install pyvisa
```