Python and Tools

Linux (Ubuntu/Mint)

Introduction

Running Linux (Ubuntu 18.04/ Mint 19 or newer) and want to use python?
You could use the default system install Python, "System Python".
On Linux Python is installed by default, when you type `python` in a terminal, a reply is showed as:

```
Python 3.8.0 (default, Nov  6 2019, 18:38:15)
[GCC 7.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Type `exit()` to exit the interpreter.

So, why not use it and forget all possible popping up issues and problems when installing different versions of Python? The Python installed by default really *belongs* to the operating system. It came installed with the operating system and is used by the system or tools installed on the system.

Using that Python version is very well possible but: For projects of your own possibly libraries and packages need to be installed and those can/might/will interfere, have impact on system tools using the default installed Python.
Another reason not to use the default installed Python is that it is installed in system directories. System directories belong to the root or sudo user and that will cause all kind of permission problems.

Thus let us install our own version of Python, but install it such a way that it's easy to manipulate, upgrade, and ... In order to be able to do this a "Package manager" is needed. Package Managers are the software tools like **apt** for Ubuntu and derivatives. One of the package managers dealing with Python is ***pyenv***. This tool allows one to

1. Install Python in user space
2. Install and use multiple versions of Python
3. Specify the exact Python version wanted for a project.
4. Switch between the installed versions
5. Use multiple versions of Python on one system.
6. ... (much more).

Let's install *pyenv* and walk through it

Remark:

pyenv is also a tool allowing to use virtual environments (above points 4 and 5 benefit from that).
This is something for later on or read about it on the *pyenv* pages on the WWW.
The goal of this document is to use *pyenv* to easily run and use python and to easily maintain it.

Install Pyenv

- Before being able to flawless install *pyenv* some build dependencies need to be installed. Installing *pyenv* means building it up from source. That process needs specific tools installed in the OS.
- Run below in a terminal for all *pyenv* Ubuntu/Mint build dependencies

```
sudo apt install -y make build-essential libssl-dev zlib1g-dev \
libbz2-dev libreadline-dev libsqlite3-dev wget curl llvm libncurses5-dev \
libncursesw5-dev xz-utils tk-dev libffi-dev liblzma-dev python-openssl
```

- When above dependencies are installed it's time to install *pyenv* itself.
  Run below in a terminal (*pyenv* and Python are installed as user, so no need for sudo anymore):

  ```
  curl https://pyenv.run | bash
  ```

  Pyenv will be installed with a few useful plugins:

  1. **`pyenv`**: The actual `pyenv` application
  2. **`pyenv-update`**: Plugin for updating `pyenv`
  3. **`pyenv-doctor`**: Plugin to verify that `pyenv` and build dependencies are installed
  4. **`pyenv-which-ext`**: Plugin to automatically lookup system commands
  5. **`pyenv-virtualenv`**: Plugin for `pyenv` and virtual environments

  Finally the install will show something like this in the terminal

```
WARNING: seems you still have not added 'pyenv' to the load path.
# Load pyenv automatically by adding
# the following to ~/.bashrc:
export PATH="$HOME/.pyenv/bin:$PATH"
eval "$(pyenv init -)"
eval "$(pyenv virtualenv-init -)"
```

Edit the `.bashrc` file in your home directory or edit/create a `.bashrc_aliases` file.
Add the lines, as given above, to one of the files or modify the lines a bit to fit a higher purpose.

```
# Pyenv Python installer, Python and other Python stuff goes below
export PYENV_ROOT=$HOME/.pyenv
export PATH=$PYENV_ROOT/bin:$PATH
eval "$(pyenv init -)"
eval "$(pyenv virtualenv-init -)"
```

Remark: a *.bashrc* file normally contains a command to source a *.bashrc_aliases* file when it is available. Using a *.bashrc_aliases* file is a save (and smart) solution. All user added items to a *.bashrc* file fit in this file making it easy to return to a basic setup.

As extra install the *pyenv* pip-migrate plugin. This plugin makes it easy to take libraries and packages installed with pip from one Python version into another.

```
git clone git://github.com/pyenv/pyenv-pip-migrate.git $(pyenv
root)/plugins/pyenv-pip-migrate
```

- Pyenv knows a lot of commands, the most used are listed here:
  - help Display help for a command
  - version Show the current Python version and its origin
    --version Display the version of *pyenv*
  - install Install a Python version using python-build
  - uninstall Uninstall a specific Python version
  - local Set or show the local application-specific Python version
  - global Set or show the global Python version
  - migrate Migrate pip packages from a Python version to another

  For all other *pyenv* command just type `pyenv` in a terminal or command window.

Install Python_3.7 or newer

- First find out what Python versions are available. This allows that the most recent version or the version required can be installed and use. From the list of versions several can be picked and installed.
  In a terminal type: `pyenv install --list | grep " 3\.[6789]"`
  The terminal will output a list of available Python versions.
  When omitting the `grep` the result will be a very long list!
- Find the version needed and install it as: `pyenv install -v 3.8.0`
  Pyenv will install Python version 3.8.0 and the -v (verbose) will list a lot of text while installing and building Python. Remove the -v to get rid of most text lines.
- pyenv Installs each Pyhton version in its own directory under the pyenv umbrella.
  The 3.8.0 version will be installed in `./home/<user>/.pyenv/versions/3.8.0`

When a second Python version is installed, let's say 3.7.4, it will reside in `~/.pyenv/versions/3.7.4`.

- Packages installed with pip (the pip belonging to the python version) are also installed under the directory <Python_version>.

Use a installed Python version

- Type in a terminal: `pyenv versions`
- The output will most likely be something like:

```
* system (set by <path/to/the/system/python)
  2.7.15
  3.7.4
  3.8.0
```

  The ***** shows what Python version is in use.

- To use one of the just installed Python versions type: `pyenv global 3.8.0`
  Running `pyenv versions` will now have a ***** in front of 3.8.0 indicating that this is now the version used by you as user.
- To check if the installation went OK and is working fully type `python -m test` in the terminal.
  This will run a long list of Python checks and test, assuring at the end everything works OK.
  This command will take a long time as a lot of tests are run (interrupt with CTRL-C).
- Everything is ready now to start using Python.
  But better first install some packages that make a programmers life easier.
  Go to **Python Libraries/Packages**.

Microsoft

Introduction

Python is not part of the windows OS and therefor it can be installed as any other application or tool for Windows. There is no real need for a package manager like *pyenv* although *pyenv* can be very useful on Windows too.

Install Python_3.7 or newer

Remark: DON'T use Python 2.x anymore please!

- Download the 64-bit version of python from the python website.
  - Downloaded the 64-bit web install version.
- Save the download somewhere on disk.
- Double click the downloaded exe file.
- A python installer launches and proposes to “install Now”.
  - When you agree to this python will be installed in C:\Users<Your_User_Name.....
  - When you tick -> Customise installation
    - Optional Features can be installed (all are selected, leave it like that).
    - Click [Next] to get to the next option window.
    - Tick:

        - Associate files with Python
        - Create shortcuts for installed applications
        - Add Python to the environment variables
        - Set the path where Python should be installed.
            - For me that's: C:\CaeTools\Python
    - Hit [Install] and wait and finally tick [Close]

Upgrade a Python installation

Every minor version of Python, that is any 3.x version, will install side-by-side with other versions on your computer. Only patch versions will upgrade existing installations.

As for PyPI packages, every Python installation comes with its own folder where modules are installed into. So if you install a new version and you want to use modules you installed for a previous version, you will have to install them first for the new version.

Python libraries/packages

Python add-on libraries or packages are there to make life easier.
When in need for something, in a lot of cases somebody already developed it.
Those add-ons can be download from the PyPi website. PyPi is a huge database/repository of community developed and shared software utilities.

PyPi packages are installed in a terminal/command window using **pip** (**pip** works the same for Linux and Microsoft).

Remark:

Using a Python version allows that a set of packages/libraries for/with that version are installed and used. The packages are installed in the same *pyenv* directory containing the Python version.

When a new Python version is installed all in a previous version installed packages are gone. The remain installed with the old Python version.

To get all previous used packages back into the new Python version use the **pyenv-pip-migrate** tool.
`pyenv migrate <old_version> <new_version>` Like: *pyenv migrate 3.7.4 3.8.0*

For our needs we will install some:

Jupyter

- The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualisations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modelling, data visualisation, machine learning, and much more. The next generation of Jupyter is called JupyterLabs and can be installed the same way as the classic Jupyter.
- The jupyter web site with documentation, tutorial and more can be found here:
    - https://jupyter.org
- ```
  pip install --upgrade pip
  pip install jupyter
  or
  pip install jupyterlab
  ```
  ```

Remark: Jupyter notebook used with Atom

When Github Atom editor is installed and used, download and install the Hydrogen package.

- Launch Atom
- Click in the top menu bar [Packages] - [Settings View] - [Install Packages/Themes]
- Type in the entry bar under [Install Packages]: Hydrogen and click the [Install] button.
- Configure the package afterwards by clicking the [Settings] button.

Spyder

- Spyder is a powerful scientific environment written in Python, for Python, and designed by and for scientists, engineers and data analysts. It offers a unique combination of the advanced editing, analysis, debugging, and profiling functionality of a comprehensive development tool with the data exploration, interactive execution, deep inspection, and beautiful visualization capabilities of a scientific package.

  Beyond its many built-in features, its abilities can be extended even further via its plugin system and API. Furthermore, Spyder can also be used as a PyQt5 extension library, allowing developers to build upon its functionality and embed its components, such as the interactive console, in their own PyQt software.

- Web site:
  - https//www.spyder-ide.org
- Before installation these dependencies MUST be installed.
  - Python 2.7 or >=3.3 DO NOT USE Python 2 ANYMORE, please!
  - PyQt5 >=5.5
  - Qtconsole >=4.2.0 – for an enhanced Python interpreter.
  - Rope >=0.9.4 and Jedi >=0.9.0 – for code completion, go-to-definition and calltips in the Editor.
  - Pyflakes – for real-time code analysis.
  - Sphinx – for the Help pane rich text mode and to get our documentation.
  - Pygments >=2.0 – for syntax highlighting and code completion in the Editor of all file types it supports.
  - Pylint – for static code analysis.
  - Pycodestyle – for style analysis.
  - Psutil – for memory/CPU usage in the status bar.
  - Nbconvert – to manipulate Jupyter notebooks on the Editor.
  - Qtawesome >=0.4.1 – for an icon theme based on FontAwesome.
  - Pickleshare – To show import completions in the Editor and Consoles.
  - PyZMQ – To run introspection services in the Editor asynchronously.
  - QtPy >=1.2.0 – To run Spyder with different Qt bindings seamlessly.
  - Chardet >=2.0.0– Character encoding auto-detection in the Editor.
  - Numpydoc Used by Jedi to get return types for functions with Numpydoc docstrings.
  - Cloudpickle Serialize variables in the IPython kernel to send them to Spyder.

    Optional Modules

  - Matplotlib >=1.0 – for 2D and 3D plotting in the consoles.

    - Pandas >=0.13.1 – for viewing and editing Series and DataFrames in the Variable Explorer.
    - Numpy – for viewing and editing two or three dimensional arrays in the Variable Explorer.
    - Sympy >=0.7.3 – for working with symbolic mathematics in the IPython console.
    - Scipy – for importing Matlab workspace files in the Variable Explorer.
    - Cython >=0.21 – to run Cython files or Python files that depend on Cython libraries in the IPython console.
- Install now Spyder
    - ```
      pip install spyder
      ```
- After installation, launch Spyder and customise it to your needs.
- Something that might be useful in Spider and when installed in Atom too is Kite.

  Kite
    - Kite is a plugin for Spyder and also for Atom.
    - **Kite** is a free AI-powered autocomplete for **Python** developers. Code faster with the **Kite** plugin for your code editor, featuring Intelligent Snippets, Line-of-Code Completions, **Python** docs, and cloudless processing.
    - Read also this article before you start:
        - https://qz.com/1043614/this-startup-learned-the-hard-way-that-you-do-not-piss-off-open-source-programmers/
    - Web site:
        - https://kite.com/
    - Install Kite
        - When Spyder is launched the first time, it is possible that a popup appears asking to install Kite. Thus, just follow the installation procedure.
        - If no popup appear or not using Spyder but Atom and want Kite, go to the Kite web site and follow the install procedure.

Numpy

- NumPy is the fundamental package for scientific computing with Python. It contains among other things:
    - A powerful N-dimensional array object.
    - Sophisticated (broadcasting) functions.
    - Tools for integrating C/C++ and Fortran code.
    - Useful linear algebra, Fourier transform, and random number capabilities.
      Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.
- The numpy web site with documentation and more can be found here:
    - https://numpy.org
- ```
  pip install numpy
  ```

Matplotlib

- Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and |Python shells, the |: yr notebook, web application servers, and four graphical user interface toolkits.
- Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, errorcharts, Scatterplots, ¢ etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery For simple plotting the pyplot module rovides : a , MATLAB- like interface, “particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.
- The matplotlib documentation can be found here:
    - https://matplotlib.org
- ```
  pip install matplotlib
  ```

Pandas

- Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with structured (tabular, multidimensional, potentially heterogeneous) and time series data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python.
- Pandas is well suited for many different kinds of data:
    - Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet.
    - Ordered and unordered (not necessarily fixed-frequency) time series data.
    - Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels
    - Any other form of observational / statistical data sets. The data actually need not be labelled at all to be placed into a pandas data structure.
- ```
  pip install pandas
  ```

Above tools are all installed to make working with *MyHdl* easy. *Jupyter* allows one to write code and document it. Myhdl can create from a described hardware system verilog and/or vhdl files so that the system can be implemented in an FPGA using Vivado (Synthesis and implementation). On other side the output of MyHdl can be any other format that can be created by using python. Therefore the *numpy, matplotlib* and *pandas* python libraries can be used. Extra waveform help for those working with Jupyter notebook is offered with the MyHdlPeek library.

MyHdl

- MyHDL turns Python into a hardware description and verification language, providing hardware engineers with the power of the Python ecosystem.
- The MyHdl documentation can be found here:
    - https://myhdl.org
    - https://doc.myhdl.org/en/stable
- ```
  pip install myhdl
  ```

MyHdlPeek
  ```

- Why should or can you use this.
- In most designs the test bench is written out as a VCD file. This file format can be viewed by a waveform viewer as GtKwave, a free available tool for waveform viewing. The VCD file can also be displayed in the waveform viewer of Vivado[^1] or in Mentor Modelsim/Questasim[^2] when one or all of those are available or in other third party waveform viewers [^3][^4].
- MyHdlPeek provides a waveform viewer in the Jypiter notebook so that documenting notebooks with waveforms no longer require screen captures of waveform viewer windows. MyHdlPeek is a module that lets you monitor signals in a MyHDL digital system simulation and display them as wave-forms in a Jupyter notebook.
- Myhdlpeek implements a Peeker object that monitors a signal and records the time and value when it changes. Just add multiple Peekers where you want to monitor something (even at sub-levels of a hierarchical design) and then view the collected timing wave-forms with a single command. You can also select which signals are shown, set the beginning and ending times of the display, and set other options.
- Documentation can be found here:
    - https://xesscorp.github.io/myhdlpeek/docs/_build/singlehtml/index.html
- ```
  pip install myhdlpeek
  ```

Below are two bonus Python libraries one can install to do more than just hardware system design.
Pyside2 allows the use of QT for generating professional looking GUI tools and Urubu allows quick generation of static web pages.

PySide2

- Qt for Python is the project that provides the official set of Python bindings (PySide2) that will supercharge your Python applications. While the Qt APIs are world renowned, there are more reasons why you should consider Qt for Python. The first official release of the PySide2 module is available now! But what is Qt (Qute)?
- Qt is a platform to create high-performance, intuitive (if they are designed to be) UI, applications and other for embedded, desktop and all kind of other platforms. Qt appears in our homes, our cars, our workplaces and our pockets smart devices. Qt makes sure you can create unique & modern connected devices that stand out from the crowd.

Installation 1:

- Open a web browser and go to: https://download.qt.io/snapshots/ci/pyside/5.13/latest/pyside2/
- Download the latest version of PySide2 for the OS of the computer it needs to run on. At the time of writing of this text it was: PySide2-5.13.1a1.dev1567626422-5.13.1-cp35.cp36.cp37-none-win_amd64.whl
    - Save the downloaded wheel file.
- Open a command window and type:

  ```
  pip install <drive>:/<Path>/To/Downloaded/wheel/File
  ```

    - Test if everything went well pip show pyside2

Installation 2:
  ```

- Documentation on PyPi can be found here: https://pypi.org/project/PySide2

```
pip install pyside2
```

Remark 1:

- To do stuff with PySide, the Qt tools must be installed. That is subject for another text.
- Qt (cute) is basically an aid to give to C/C++ easy graphical possibilities.
- Due to that PySide2 has dependencies to a C/C++ parser clang.

Remark 2:

- There is not so much information available on teh use of Python with Pyside. I found a good web page that shows two possible uses of Qt and Python with Pyside2.
- https://doc.qt.io/qtforpython/tutorials/basictutorial/uifiles.html

Urubu

- A micro CMS for static websites
- Goto this page for all documentation.
- Install Urubu from the command line: pip instal] urubu
- There is a quickstart tool that lowers the learning cost of Urubu. The quickstart tool is at the same time a kind of tutorial. To get it, download the quikstart zip file from the Urubu website.

The end for now.

Marc Defossez

07 Feb 2020