Introduction

A *Python* installation basically consists in:

- the *Python* executable itself: `bin` directory
- the default packages included within Python
- the third party packages installed by the user: `lib/*/site-packages` directory

Whenever one installs a new package, others are updated first and then it is downloaded and installed.

Thus, there exists a risk to break the compatibility with previous packages. Moreover, default Python installation is - regarding Linux - a system-wide installation: if some components are modified there exists some instability risks.

For these reasons, the preferred way is to create an environment as soon as a new Python project is started. The simplest way is to use environments managers such as ***Anaconda***.

Anaconda

Loading [MathJax]/extensions/Safe.js

Introduction

Anaconda is a software that manages Python environments. The software comes with a graphical interface (Anaconda Navigator) but the preferred way is to use the command line: faster, more stable. The document is accessible here.

Hereafter, all commands must be ran in a command line.

Create an environment

the first step is to install Anaconda (or Miniconda, a smaller alternative). Once installed, an environment can be created using a command similar to this one:

```
conda create -n myenv python=3.11 scipy=0.17.3 astroid babel
```

We can notice :

- the environment name ('myenv')
- the Python version (3.11)
- packages to be installed, with some specified versions

Instead of specifying a name, one can specify a location of the new environment:

```
conda create --prefix ./envs python=3.11
```

Activate an environment

Activating an environment is telling the computer that everything that relates to Python must be ran within the environment.

Using the command line

In a command prompt, `conda info --envs` gives a list of available environments.

A star `*` shows the currently activated environment. By default, it is the 'base' environment, i.e the one that comes with Anaconda installation. The 'base' environment must **never be used**.

Let's activate `myenv`:

```
conda activate myenv
```

'myenv' is showed in parenthesis (instead of 'base').

Once activated, we can manage this environment:

- list current packages: `conda list`.
- install new packages: `conda install` (complete documentation).

Loading [MathJax]/extensions/Safe.js

We can also:

- start a new interactive session:
    - `python` is the standard Python interpreter
    - `ipython` if the magic Python interpreter, with extended commands and friendly interface (install needed)
- run a Pyhon file: `python my_program.py`.

Using Vscode

If the environment is not automatically detected, it can be chosen using the `Select interpreter` interpreter tool (using `Ctrl` + `Maj` + `P`). In this tool, one can specify the executable Python path (`bin` directory of the environment).

Duplicate an environment

Environment duplication makes it possible to work in the same conditions on several different computers (for instance, a personnal computer and a working station). Yet, it works best when all computers have the same operating system.

From computer 1 ...

Once the environment is activated:

```
conda list --explicit > spec-file.txt
```

This command exports to a file (`spec-file.txt`) all the details of the environment.

... to computer 2

```
conda create --name same_env_computer_2 --file spec-file.txt
```

This command install an environment following specifications of `spec-file.txt`.

Loading [MathJax]/extensions/Safe.js

Notes

- Anaconda is not only a Python packages manager: a conda environment can handle other softwares and successfully isolate them from the remaining of the operating system.

- Regarding Python, only some packages can be installed using Anaconda:

    1. some packages exist only on the official Python package repository, called PyPi. Those must be installed using `pip`.

       In a conda environment, it is strongly unrecommend to mix conda and pip packages (though it's possible). The prefered procedure is:

        - whenever it's possible, install the conda version of the package
        - install the pip version when there is no other choice
        - try to avoid conda packages after some pip packages were installed

    2. some packages exist only as source code that can be retrieved from shared plateforms such as GitHub or GitLab

- Anaconda is not the only way to handle Python environments. Another way is using the `venv` module. Pros include:

    - compatible with pip
    - lighter than conda

  Yet, managing venv environments is slightly less intuitive than conda environments.