

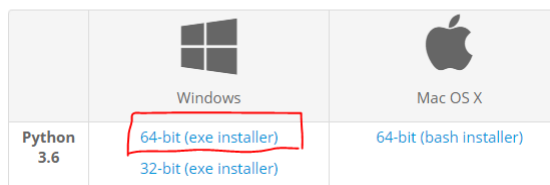
Getting ready with python, virtual environments, pyomo and jupyter

I) Install Python with the Miniconda distribution

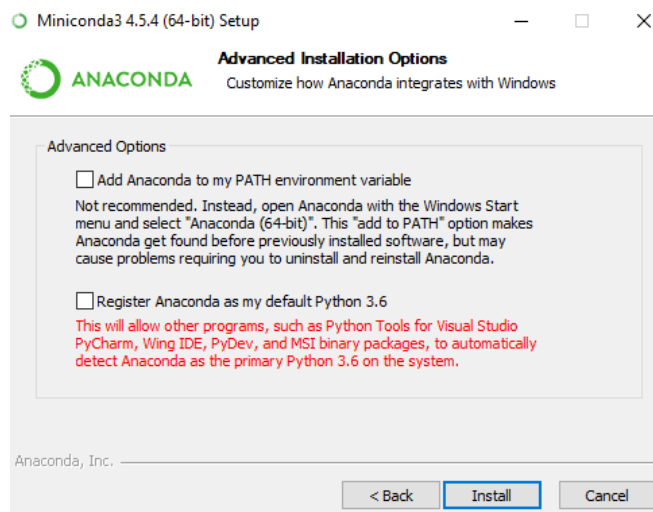
First we need to install python on our system. The heart of the language is the **python interpreter**, which runs our code. We will download this interpreter as part of a distribution called Miniconda, which provides an excellent package manager called conda (next section). Miniconda is the lightweight brother of the famous scientific computing distribution Anaconda.

1. Go to <https://conda.io/miniconda.html>
2. Download the installer for Python 3 that is suitable for your operating system. In the course we work with 64-bit windows machines.

Miniconda



3. Install Miniconda with the following options:
 - Install for: Just me.
 - Do not alter suggested installation path.
 - For this course: **Uncheck** both checkboxes in the advanced options.

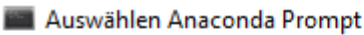
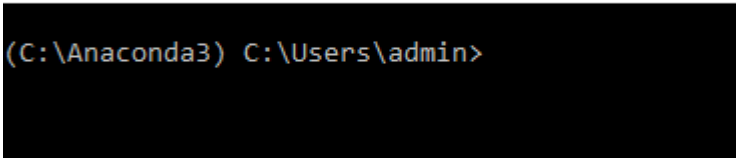


II) Getting to know the package manager conda

In the following we will get to know a tool provided by the Miniconda distribution called **conda**.

"Conda is an open source package management system and environment management system that runs on Windows, macOS and Linux. Conda quickly installs, runs and updates packages and their dependencies. Conda easily creates, saves, loads and switches between environments on your local computer. It was created for Python programs, but it can package and distribute software for any language." -<https://conda.io/docs/>

We are going to use conda for the following tasks:

- Setting up a virtual environment (explained in the next section).
 - Installing modules (libraries) that we need in our project.
 - Running Jupyter Notebook, a program that allows us to interactively write and execute python code in a very informative and user-friendly way.
-
1. In the Windows Search, type "Anaconda Prompt" and open it. A command window will appear similar to the image below.


 2. The first part enclosed by brackets () is your current virtual environment you are in (more about that in the next part). The second part is the path to the current working directory.
 3. Type `conda` and have a quick look at the output. There is a lot of text, but there are also some interesting commands on the left: info, list, create, install, upgrade, remove... etc.
 4. Before we go on, let's make sure our conda is up to date by typing `conda update conda`. If you are asked for permission, confirm by pressing the y key and then return.

III) Set up a virtual environment

First of all, what are virtual environments and why do we use them?

Python applications will often use packages and modules that don't come as part of the standard library. Applications will sometimes need a specific version of a library, because the application may require that a particular bug has been fixed or the application may be written using an obsolete version of the library's interface.

This means it may not be possible for one Python installation to meet the requirements of every application. If application A needs version 1.0 of a particular module but application B needs version 2.0, then the requirements are in conflict and installing either version 1.0 or 2.0 will leave one application unable to run.

The solution for this problem is to create a virtual environment, a self-contained directory tree that contains a Python installation for a particular version of Python, plus a number of additional packages.

Different applications can then use different virtual environments. To resolve the earlier example of conflicting requirements, application A can have its own virtual environment with version 1.0 installed while application B has another virtual environment with version 2.0. If application B requires a library be upgraded to version 3.0, this will not affect application A's environment. - <https://docs.python.org/3/tutorial/venv.html>

In the next steps we will setup a virtual environment called **pyomo_tutorial**. It will contain a specific version of the python interpreter (3.6.5) as well as a bunch of packages that let us work through the optimization examples of the SAMO course. Even in a few years, you can activate this virtual environment and everything will work together the same way it does now, no matter how python and the whole package ecosystem developed. That is the real strength of virtual environments. If you are interested you can find extensive information about virtual environments [here](#).

1. Let's first check what virtual environments already exist. Type `conda info --envs`
You should see a (very short) list that only contains one conda environment, namely the root we are currently in. You should recognize its name, as it is the same as in the brackets left of your command line cursor.
2. Now create a new virtual environment with the name "pyomo_tutorial" and python version 3.6.5. It's easy! Type `conda create --name pyomo_tutorial python=3.6.5`
Conda will show you some packages that will be installed and ask for permission to proceed.

```
The following NEW packages will be INSTALLED:

certifi:          2016.2.28-py36_0
pip:              9.0.1-py36_1
python:           3.6.5-h0c2934d_0
setuptools:       36.4.0-py36_1
vc:               14.1-h0510ff6_3
vs2015_runtime:   15.5.2-3
wheel:            0.29.0-py36_0
wincertstore:     0.2-py36_0

Proceed ([y]/n)? y
```

3. Type again `conda info --envs`
You should now see your new environment with name pyomo_tutorial in the list.
Congratulations to your first virtual environment!

```
(C:\Anaconda3) C:\Users\admin>conda info --envs
# conda environments:
#
pyomo_tutorial      C:\Anaconda3\envs\pyomo_tutorial
root                * C:\Anaconda3
```

4. Enter the newly created environment by typing `activate pyomo_tutorial`
Notice how the name in the brackets changes to (pyomo_tutorial). You are now in the virtual environment pyomo_tutorials, which is isolated from the rest of your system.
5. Type `python` to activate python for a test. Note that your command cursor changes to `>>>`.
6. Type `1+1` and `print('This is Python!')`
These are python statements that are executed by the python interpreter within your virtual environment. To see where your python interpreter is located, type `import sys` followed by `sys.executable`
You get back a path that points to the current python interpreter. Note how it is located within your virtual environment named pyomo_tutorial.
7. Exit python by typing `quit()`

Note how the cursor changes to the old style: (name_of_environment)
path\to\current\working\directory

```
(pyomo_tutorial) C:\Users\admin>python
Python 3.6.5 [Anaconda, Inc.] (default, Mar 29 2018, 13:3
) on win32
Type "help", "copyright", "credits" or "license" for more
>>> 1+1
2
>>> quit()
(pyomo_tutorial) C:\Users\admin>
```

IV) Installing the necessary python packages

One strength of python is the huge amount of code that many people have written over the years and that we can easily use for our purpose. We will make use of large packages like pyomo for optimization, geopandas for manipulating shapefiles and numpy for data processing. These packages themselves make use of dozens of smaller packages (dependencies). As it would be impossible to track down and install each dependency on our own, we will make use of two package managers that take care of most of the work: pip and conda.

Package managers are the way how we easily extend the functionality of our python. Need R-like statistical methods? Install the pandas package. Want to create a deep neural network? Install the keras package. Want Matlab-like array processing power? Install the numpy package.

Although the python community tries its best to keep everything nice and simple, this is not always possible in such a complex and decentralized system. You will notice in the next steps, that some packages need some workarounds to really work. This might seem confusing and like a disadvantage when coming from proprietary languages like Matlab where MathWorks makes sure everything works together fine. It is certainly true that it may take some time to figure out how everything works together, but in my opinion this is a comparatively small price to pay for the extensive and free package ecosystem python provides. In addition, this point underlines the importance of virtual environments to “freeze” a configuration of packages that works.

In the following we use pip to install some packages and conda for others. For this course it is sufficient to say that you should first try to install with the conda command and if this fails follow up with pip. If you are looking for more in-depth explanation on the differences of conda and pip, start [here](#).

1. Make sure you are in your virtual environment.

```
(pyomo_tutorial) C:\Users\admin>
```

2. Make sure your version of pip is up to date. Type
`python -m pip install --upgrade pip`

3. You can list all the packages in your current virtual environment by typing `conda list`
The current list of packages in your newly created environment should be quite short.

4. Install pyomo, which is the optimization framework we will use. Type:

```
pip install pyomo==5.5.0
```

Note: I will use specific version numbers to make sure everything works the same as when I tested it. Omitting the version number gives you the most recent version and is usually a good way to go.

5. Install geopandas, which allows us to process shape files in a userfriendly way. Geopandas is based on the pandas package, which offers R-like dataframe and statistics capabilities. This is a huge beast and may take some time to install. The creators [recommend](#) to install geopandas from the conda-forge channel, that's why we add the parameter "-c conda-forge". Type:

```
conda install -c conda-forge geopandas=0.4.0
```

- If you install geopandas on your private computer, you may get an error that "Microsoft Visual C++ 14.0 is required". Some dependencies of geopandas rely on this microsoft compiler and you need to install it manually. A good starting point in that case is [this page](#).

6. At the time of writing, the two geopandas dependencies **Fiona** and **GDAL** do not correctly work together on windows systems (see [here](#)). Luckily, a guy named Christoph Gohlke presents on his [page](#) slightly modified versions of these packages that work together. You can download them as wheel files. Think of wheel files (*.whl) just like of zip files. **The files are provided in the folder "Fiona_Gdal_wheels"**. Select the ones with "win32" or "amd64" in their name based on whether you work on a 32-bit or 64-bit machine respectively. The packages are installed like any other package using pip by providing the full path to the wheel file. However, because we already installed a version of fiona and gdal during the installation of geopandas, we need to **force pip to reinstall** and therefore overwrite these packages. To that end, type

```
pip install --upgrade --force-reinstall path_gdal
```

whereby path_gdal is the full path to the gdal wheel file including the file (*.whl).

7. Repeat the command with the path to the fiona wheel:

```
pip install --upgrade --force-reinstall path_fiona
```

whereby path_fiona is the full path to the fiona wheel file including the file (*.whl).

8. The following package is needed to read Excel files. Type:

```
conda install xlrd=1.1.0
```

9. Install Jupyter Notebook (explained in next section), type:

```
conda install jupyter=1.0.0
```

In the course, when asked for the administrator password, click No. The installation should continue normally.

10. Test the packages you just installed! Enter python by typing: `python`

11. Import pyomo. Type `import pyomo`

If no error message appears and you get a new line with the >>> at the beginning everything went well.

12. Import geopandas. Type `import geopandas`
13. Import jupyter. Type `import jupyter`
14. If no error appeared, congratulations! Exit python by typing `quit()`

V) Setup Jupyter notebook

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more. - <http://jupyter.org/>

Jupyter notebooks are very useful, if your code produces a lot of intermediary results that need explanation or visualization. This is a common case in data science. Jupyter notebooks then provide a way to nicely document your code with formatted text and latex like formulas in combination with code blocks that can be interactively run step by step by the user.

The underlying machinery that allows code in jupyter notebooks to be interactively run is called a **kernel**, which is basically a link between the jupyter notebook and a python interpreter. We will create a kernel for our virtual environment to tell jupyter notebook to make use of the python interpreter located within the environment.

1. If you are not already within your virtual environment indicated by the round brackets left of your input cursor showing (pyomo_tutorial), activate the virtual environment with `activate pyomo_tutorial`
2. Install a new kernel for the pyomo_tutorial environment by typing `python -m ipykernel install --user --name pyomo_tutorial --display-name "Python 3.6.5 (pyomo_tutorial)"`
3. The root directory of jupyter is always your current working directory when starting jupyter. Therefore, navigate to the PYOMO Workshop folder using the cd command: `cd path\to\pyomo\workshop\folder`
4. Start the jupyter notebook by typing: `jupyter notebook`
5. Now your browser should open. Click **New** in the top right corner to create a new jupyter notebook. If all went well, in the dropdown menu you should be able to select the kernel named "Python 3.6.5 (pyomo_tutorial)" that you just created. Now you created your first jupyter notebook!