SoftPython

Introductive guide to data cleaning and analysis with Python 3

David Leoni, Alessio Zamboni, Marco Caresia

Sep 22, 2020

CONTENTS

Prefazione

Introductive guide to coding, data cleaning and analysis for Python 3, with many worked exercises.

WARNING: THIS ENGLISH VERSION IS IN-PROGRESS, COMPLETION IS DUE BY END OF 2020

ITALIAN VERSION IS HERE: it.softpython.org[1]

Nowadays, more and more decisions are taken upon factual and objective data. All disciplines, from engineering to social sciences, require to elaborate data and extract actionable information by analysing heterogenous sources. This book of practical exercises gives an introduction to coding and data processing using Python[2], a programming language popular both in the industry and in research environments.

News

Old news: link

[1] https://it.softpython.org
[2] https://www.python.org

CHAPTER

ONE

PANORAMICA

1.1 Intended audience

This book can be useful for both novices who never really programmed before, and for students with more techical background, who a desire to know about about data extraction, cleaning, analysis and visualization (among used frameworks there are Pandas, Numpy and Jupyter editor). We will try to process data in a practical way, without delving into more advanced considerations about algorithmic complexity and data structures. To overcome issues and guarantee concrete didactical results, we will present step-by-step tutorials.

1.2 Contents

Overview

- Approach and goals
- Resources

1.2.1 A - Foundations

1.3 Authors

David Leoni (main author): Software engineer specialized in data integration and semantic web, has made applications in open data and medical in Italy and abroad. He frequently collaborates with University of Trento for teaching activities in various departments. Since 2019 is president of CoderDolomiti Association, where along with Marco Caresia manages volunteering movement CoderDojo Trento to teach creative coding to kids. Email: david.leoni@unitn.it Website: davidleoni.it[3]

Marco Caresia (2017 Autumn Edition assistent @DISI, University of Trento): He has been informatics teacher at Scuola Professionale Einaudi of Bolzano. He is president of the Trentino Alto Adige Südtirol delegatioon of the Associazione Italiana Formatori and vicepresident of CoderDolomiti Association.

Alessio Zamboni (2018 March Edition assitent @Sociology Department, University of Trento): Data scientist and software engineer with experience in NLP, GIS and knowledge management. Has collaborated to numerous research projects, collecting experinces in Europe and Asia. He strongly believes that *'Programming is a work of art'*.

Massimiliano Luca (2019 summer edition teacher @Sociology Department, University of Trento): Loves learning new technilogies each day. Particularly interested in knowledge representation, data integration, data modeling and computational social science. Firmly believes it is vital to introduce youngsters to computer science, and has been mentoring at Coder Dojo DISI Master.

[3] https://davidleoni.it

1.4 License

The making of this website and related courses was funded mainly by Department of Information Engineering and Computer Science (DISI)[4], University of Trento, and also Sociology[5] and Mathematics[6] departments.

Technical notes: all website pages are easily modifiable Jupyter notebooks, that were converted to web pages using NBSphinx[8] using template Jupman[9]. Text sources are on Github at address https://github.com/DavidLeoni/softpython-en

1.5 Acknowledgments

We thank in particular professor Alberto Montresor of Department of Information Engineering and Computer Science, University of Trento to have allowed the making of first courses from which this material was born from, and the project Trentino Open Data (dati.trentino.it[10]) for the numerous datasets provided.

Other numerous intitutions and companies that in over time contributed material and ideas are cited in this page

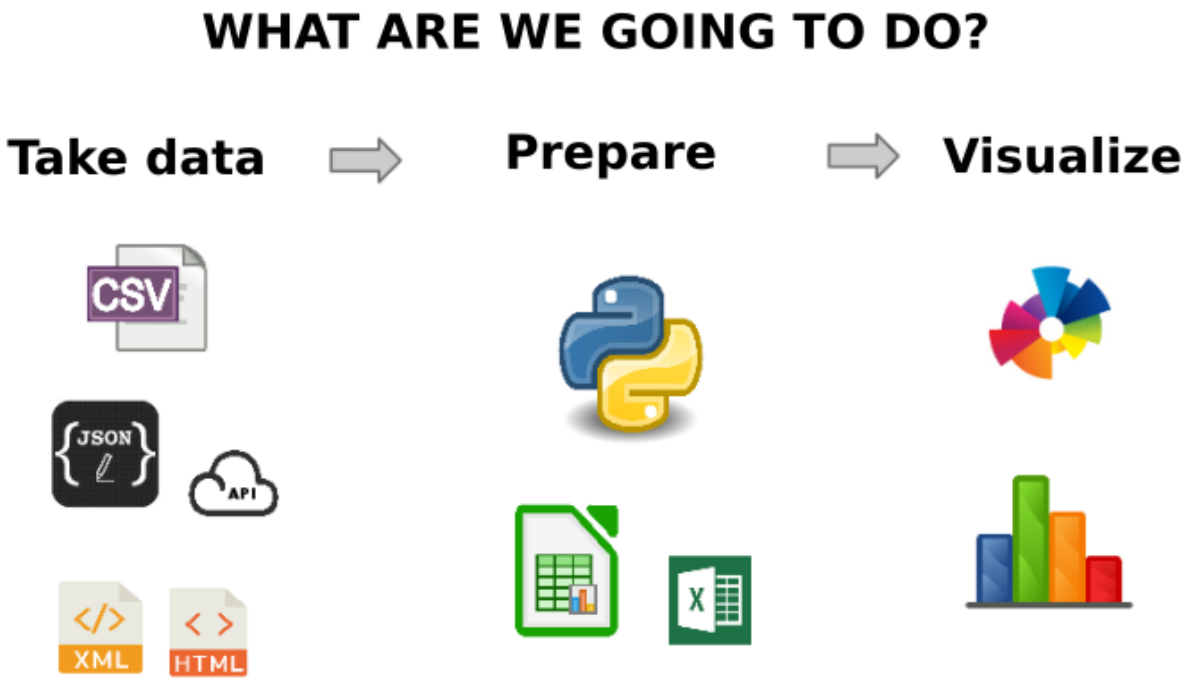
[4] https://www.disi.unitn.it
[5] https://www.sociologia.unitn.it/en
[6] https://www.maths.unitn.it/en
[7] https://it.softpython.org/index.html#Autori
[8] https://nbsphinx.readthedocs.io
[9] https://github.com/DavidLeoni/jupman
[10] https://dati.trentino.it

CHAPTER

TWO

OVERVIEW

To start with we will spend a couple of words on the approach and the goals of the book, then we will deep dive into the code.



2.1 Chapters

The tutorial mostly deal with fundamentals of PYthon 3, data analysis (intended more like raw data processing than statistics) and some applications (dashboard, database, ..)

What are *not* about:

- object oriented programming theory
- algorithms, computational complexity
- performance
  - no terabytes of data …
- advanced debugging (pdb)

- testing is only mentioned
- machine learning
- web development is only mentioned

2.2 Why Python?

- **Easy** enough to start with
- **Versatile**, very much used for
  - scientific calculus
  - web applications
  - scripting
- **widespread** both in the industry and research environments
  - Tiobe[11] Index
  - popularity on Github[12]
- **Licence** open source & business friendly[13]
  - translated: you can sell commercial products based on Python without paying royalties to its authors

[11] https://www.tiobe.com/tiobe-index/
[12] https://madnight.github.io/githut/#/pull_requests/2020/1
[13] https://docs.python.org/3/license.html

2.3 References

Altro materiale: Citiamo i seguenti due libri, entrambi dall'approccio discorsivo e gratuiti disponibili sia online che in pdf.

2.3.1 Part A Resources

2.3.2 Think Python, by Allen Downey

- Talks a lot, step by step, good for beginners
- online[14]
- printed[15]
- PDF[16]
- Interactive edition[17]

License: Creative Commons CC BY Non Commercial 3.0[18] as reported in the original page[19]

2.3.3 Dive into Python 3, by Mark Pilgrim

- More practical, contains more focused tutorials (i.e. manage XML files)
- online version[20]
- printed[21]
- zip offline[22]
- PDF[23]

Licence: Creative Commons By Share-alike 3.0[24] as reported at the bottom of book website[25]

[14] http://www.greenteapress.com/thinkpython/html/
[15] https://www.amazon.it/Think-Python-Like-Computer-Scientist/dp/1491939362/ref=sr_1_1?ie=UTF8&qid=1537163819&sr=8-1&keywords=think+python
[16] http://greenteapress.com/thinkpython2/thinkpython2.pdf
[17] https://runestone.academy/runestone/static/thinkcspy/index.html
[18] http://creativecommons.org/licenses/by-nc/3.0/deed.it
[19] http://greenteapress.com/wp/think-python-2e/
[20] http://www.diveintopython3.net/
[21] http://www.amazon.com/gp/product/1430224150?ie=UTF8&tag=diveintomark-20&creativeASIN=1430224150
[22] https://github.com/diveintomark/diveintopython3/zipball/master
[23] https://github.com/downloads/diveintomark/diveintopython3/dive-into-python3.pdf
[24] http://creativecommons.org/licenses/by-sa/3.0/
[25] http://www.diveintopython3.net/

2.4 Approach and goals

If you have troubles with programming basics:

- **Exercise difficulty**: ⊛ , ⊛⊛
- Read SoftPython - Parte A - Foundations[26]
- Other useful stuff can be found in the book 'Think Python'

If you already know how to program:

- **Exercise difficulty**: ⊛⊛⊛, ⊛⊛⊛⊛
- Read Python Quick Intro[27] and then go directly to Part B - Data Analysis
- other useful things can be found in the book **Dive into Python 3**

2.5 Doesn't work, what should I do?

While programming you will surely encounter problems, and you will stare at mystierious error messages on the screen. The purpose of this book is not to give a series of recipes to learn by heart and that always work, as much as guide you moving first steps in Python world with some ease. So, if something goes wrong, do not panic and try following this list of steps that might help you. Try following the proposed order:

1. If in class, ask professor (if not in class, see last two points).
2. If in class, ask the classmate who knows more
3. Try finding the error message on Google
    - remove names or parts too specific of your program, like line numbers, file names, variable names
    - Stack overflow[28] is your best friend
4. Look at Appendix A - Debug from the book Think Python[29]
    - Syntax errors[30]
        - I keep making changes and it makes no difference.[31]
    - Runtime errors[32]
        - My program does absolutely nothing.[33]
        - My program hangs.[34]
        - Infinite Loop[35]
        - Infinite Recursion[36]
        - Flow of Execution[37]

[26] https://en.softpython.org/index.html#A---Foundations
[27] https://en.softpython.org/quick-intro/quick-intro-sol.html
[28] https://stackoverflow.com
[29] http://greenteapress.com/thinkpython2/html/thinkpython2021.html
[30] http://greenteapress.com/thinkpython2/html/thinkpython2021.html#235
[31] http://greenteapress.com/thinkpython2/html/thinkpython2021.html#sec236
[32] http://greenteapress.com/thinkpython2/html/thinkpython2021.html#sec237
[33] http://greenteapress.com/thinkpython2/html/thinkpython2021.html#sec238
[34] http://greenteapress.com/thinkpython2/html/thinkpython2021.html#sec239
[35] http://greenteapress.com/thinkpython2/html/thinkpython2021.html#sec240
[36] http://greenteapress.com/thinkpython2/html/thinkpython2021.html#sec2241
[37] http://greenteapress.com/thinkpython2/html/thinkpython2021.html#sec242

  - When I run the program I get an exception[38]
  - I added so many print statements I get inundated with output[39]
- Semantic errors[40]
  - My program doesn't work[41]
  - ve got a big hairy expression and it doesn't do what I expect.[42]
  - ve got a function that doesn't return what I expect.[43]
  - I'm really, really stuck and I need help.[44]
  - No, I really need help.[45]

5. Gather some courage and ask on a public forum, like Stack overflow or python-forum.io - see *how to ask questions*.

2.5.1 How to ask questions

IMPORTANT

If you want to ask written questions on public chat/forums (i.e. like python-forum.io[46] DO FIRST READ the forum rules (see for example How to ask Smart Questions[47]

In substance, you are always asked to clearly express the problem circumstances, putting an explicative title to the post /mail and showing you spent some time (at least 10 min) trying a solution on your own. If you followed the above rules, and by misfortune you still find programmers who use harsh tones, just ignore them.

2.6 Installation and tools

- If you still haven't installed Python3 and Jupyter, have a look at *Installation*

2.7 Let's start !

- **If you already have some programming skill**: you can look Python quick start
- **If you don't have programming skills**: got to Tools and scripts[48]

[38] http://greenteapress.com/thinkpython2/html/thinkpython2021.html#sec243
[39] http://greenteapress.com/thinkpython2/html/thinkpython2021.html#sec244
[40] http://greenteapress.com/thinkpython2/html/thinkpython2021.html#sec245
[41] http://greenteapress.com/thinkpython2/html/thinkpython2021.html#sec246
[42] http://greenteapress.com/thinkpython2/html/thinkpython2021.html#sec247
[43] http://greenteapress.com/thinkpython2/html/thinkpython2021.html#sec248
[44] http://greenteapress.com/thinkpython2/html/thinkpython2021.html#sec249
[45] http://greenteapress.com/thinkpython2/html/thinkpython2021.html#sec250
[46] https://python-forum.io/index.php
[47] https://python-forum.io/misc.php?action=help&hid=19
[48] https://en.softpython.org/tools/tools-sol.html

CHAPTER

THREE

INSTALLATION

We will see how to install Python, additional Python libraries, Jupyter notebook and managing virtual environments.

3.1 Installing Python

There are varioues ways to install Python 3 and its modules: there is the official 'plain' Python distribution but also package managers (i.e. Anaconda) or preset environments (i.e. Python(x,y)) which give you Python plus many various modules. Once completed the installation, Python 3 contains a command `pip` (sometimes called `pip3` in Python 3), which allows to install afterwards other packages you may need.

The best way to choose what to install depends upon which operating system you have and what you intend to do with it. In this book we will use Python 3 and scientific packages, so we will try to create an environment to support this scenario.

Fast online alternatives

If you have difficulties installing and you are anxious to try Python, you can program directly online with Python 3 on repl.it[49]

Another useful resource is Python Tutor[50], which allows to execute one instruction per time while offering a useful visualization of what happens 'under the hood'.

You can also try the online demo of Jupyter[51], but it is not always available.

Whichever online environment you choose, always remember to check it is Python **3** !

Attention: before installing random stuff from the internet, read carefully this guide

We tried to make it generic enough, but we couldn't test all various cases so problems may arise depending on your particular configuration.

Attention: do not mix different Python distribution for the same version !

Given the wide variety of installation methods and the fact Python is available in laready many programs, it might be you already have installed Python without even knowing it, maybe in version 2, but we need the 3! Overlayinh several Python environments with the same version may cause problems, so in case of doubt ask somebody who knows more!

[49] https://repl.it/languages/python
[50] http://pythontutor.com/visualize.html#py=3
[51] http://try.jupyter.org

3.1.1 Windows installation

For Windows, we suggest to install the distribution Anaconda for Python 3.8[52] or greater, which, along with the native Python package manager `pip`, also offers the more generic command line package manager `conda`.

Once installed, verify it is working like this:

1. click on the Windows icon in the lower left corner and search for 'Anaconda Prompt'. It should appear a console where to insert commands, with written something like `C:\Users\David>`. NOTE: to launch Anaconda commands, only use this special console. If you use the default Windows console (`cmd`), Windows will not be able to find Python.
2. In Anaconda console, type:

```
conda list
```

It should appear a list of installed packages, like

```
# packages in environment at C:\Users\Jane\AppData\Local\Continuum\Anaconda3:
#
alabaster                 0.7.7                    py35_0
anaconda                  4.0.0                np110py35_0
anaconda-client           1.4.0                    py35_0
...
numexpr                   2.5                  np110py35_0
numpy                     1.10.4                   py35_0
odo                       0.4.2                    py35_0
...
yaml                      0.1.6                         0
zeromq                    4.1.3                         0
zlib                      1.2.8                         0
```

3. Try Python3 by typing in the Anaconda console:

```
C:>  python
```

It should appear something like:

```
Python 3.6.3 (default, Sep 14 2017, 22:51:06)
MSC v.1900 64 bit (Intel)[GCC 5.4.0 20160609] on win64
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Attention: with Anaconda, you must write `python` instead of `python3` !

If you installed Anaconda for Python3, it will automatically use the correct Python version by simply writing `python`. If you write `python3` you will receive an error of file not found !

Attention: if you have Anaconda, always use `conda` to install Python modules ! So if in next tutorials you se written `pip3 install whatever`, you will instead have to use `conda install whatever`

[52] https://www.anaconda.com/download/#windows

3.1.2 Installation Mac

To best manage installed app on Mac independently from Python, usually it is convenient to install a so called *package manager*. There are various, and one of the most popular is Homebrew[53]. So we suggest to first install Homebrew and then with it you can install Python 3, plus eventually other components you might need. As a reference, for installation we took and simplified this guide by Digital Ocean](https://www.digitalocean.com/community/tutorials/how-to-install-python-3-and-set-up-a-local-programming-environment-on-macos)

Attention: check if you already have a package manager !

If you already have installed a package manager like for example Conda (in *Anaconda* distribution), *Rudix*, *Nix*, *Pkgsrc*, *Fink* o *MacPorts*, maybe Homebrew is not needed and it's better to use what you already have. In these cases, it may be worth asking somebody who knows more ! If you already have *Conda/Anaconda*, it can be ok as long as it is for Python 3.

— 1 Open the Terminal

MacOS terminal is an application you can use to access command line. As any other application, it's available in *Finder*, navigation in *Applications* folder, and the in the folder *Accessories*. Frotm there, double click on the *Terminal* to open it as any other app. As an alternative, you can use *Spotlight* by pressing *Command* and *Space* to find the Terminal typing the name in the bar that appears.

- 2 Install Homebrew by executing in the terminal this command:

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
↪master/install)"
```

— 3 Add `/usr/local/bin` to `PATH`

In this passage with an unsettling name, once Homebrew installation is completed, you will make sure that apps installed with Homebrew shall always be used instead of those Mac OS X may automatically select.

— 3.1 Open a new Terminal.

— 3.2 From within the terminal, digit the command

```
ls -a
```

You will see the list of all files present in the home folder. In these files, verify if a file exists with the following name: `.profile` (note the dot at the beginning):

- If it exists, go to following step
- If it doesn't exist, to create it type the following command:

```
touch  $HOME/.profile
```

— 3.3 Open with text edit the just created file `.profile` giving the command:

```
open -e  $HOME/.profile
```

— 3.4 In text edit, add to the end of the file the following line:

```
export PATH=/usr/local/bin:$PATH
```

[53] https://brew.sh/

— 3.5 Save and close both Text Edit and the Terminal

— 4 Verify Homebrew is correctly installed, by typing in a new Terminal:

```
brew doctor
```

If there aren't updates to do, the Terminal should show:

```
Your system is ready to brew.
```

Otherwise, you might see a warning which suggest to execute another command like `brew update` to ensure the Homebrew installation is updated.

— 5. Install python3 (Remember the '3' !):

```
brew install python3
```

Along with python 3, Homebrew will also install the internal package manager of Python `pip3` which we will use in the following.

— 6 Verify Python3 is correctly installed. By executing this command the writing `"/usr/local/bin/python3"` should appear:

```
which python3
```

After this, try to launch

```
python3
```

You should see something similar:

```
Python 3.6.3 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609] on mac
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

To exit Python, type `exit()` and press Enter.

3.1.3 Linux installation

Luckily, all Linux distributions are already shipped with package managers to easily install applications.

- If you have Ubuntu:
  1. follow the guide of Dive into Python 3, chapter 0 - Installare Python[54] in particular by going to the subsection installing in Ubuntu Linux[55]
  2. after completing the guide, install also `python3-venv`:

     ```
     sudo apt-get install python3-venv
     ```

- If you *don't* have Ubuntu, read this note[56] and/or ask somebody who knows more.

To verify the installation, try to run from the terminal

[54] https://diveintopython3.problemsolving.io/installing-python.html
[55] https://diveintopython3.problemsolving.io/installing-python.html#ubuntu
[56] https://diveintopython3.problemsolving.io/installing-python.html#other

```
python3
```

You should see something like this:

```
Python 3.6.3 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

3.2 Installing packages

You can extend Python by installing several free packages. The best way to do it varies according to the operating system and the installed package manager.

ATTENTION: We will be using *system commands*. If you see `>>>` in the command line, it means you are inside Python interpreter and you must first exit: to do it, type `exit()` and press Enter.

In what follows, to check if everything is working, you can substitute `PACKAGENAME` with `requests` which is a module for the web.

If you have Anaconda:

- click on Windows icon in the lower left corner and search `Anaconda Prompt`. A console should appear where to instert commands, with something written like `C:\Users\David>`. (NOTE: to run commands in Anaconda, use only this special console. If you use the default Windows console (cmd), Windows, will not be able to find Python)
- In the console type `conda install PACKAGENAME`

If you have Linux/Mac open the Terminaland give this command (`--user` install in your home):

- `python3 -m pip install --user PACKAGENAME`
- **NOTE:** If you receive errors which tell you the command `python3` is not found, remove the `3` after `python`

INFO: there is also a system command `pip` (or `pip3` according to your system). You can directl call it with `pip install --user PACKAGENAME`

Instead, we install instead with commands like `python3 -m pip install --user PACKAGENAME` for uniformity and to be sure to install packages for Python 3 version

---

3.3 Jupyter Notebook

3.3.1 Run Jupyter notebook

A handy editor you can use for Python is Jupyter[57]:

- If you installed Anaconda, you should already find it in the system menu and also in the Anaconda Navigator.
- If you didn't install Anaconda, try searching in the system menu anyway, maybe by chance it was already installed

[57] http://jupyter.org/

- If you can't find it in the system menu, you may anyway from command line

Try this:

```
jupyter notebook
```

or, as alternative,

```
python3 -m notebook
```

ATTENTION: `jupyter` is NOT a Python command, it is a *system* command.

If you see written >>> on command line it means you must first exit Python insterpreter by writing 'exit()` and pressing Enter !

ATTENTION: If Jupyter is not installed you will see error messages, in this case don't panic and *go to installation*.

A browser should automatically open with Jupyter, and in the console you should see messages like the following ones. In the browser you should see the files of the folders from which you ran Jupyter.

If no browser starts but you see a message like the one here, then copy the address you see in an internet browser, preferebly Chrome, Safari or Firefox.

```
$ jupyter notebook
[I 18:18:14.669 NotebookApp] Serving notebooks from local directory: /home/da/Da/prj/
↪softpython/prj
[I 18:18:14.669 NotebookApp] 0 active kernels
[I 18:18:14.669 NotebookApp] The Jupyter Notebook is running at: http://localhost:
↪8888/?token=49d4394bac446e291c6ddaf349c9dbffcd2cdc8c848eb888
[I 18:18:14.669 NotebookApp] Use Control-C to stop this server and shut down all␣
↪kernels (twice to skip confirmation).
[C 18:18:14.670 NotebookApp]

    Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
        http://localhost:8888/?token=49d4394bac446e291c6ddaf349c9dbffcd2cdc8c848eb888
```

ATTENTION 1: in this case the address is `http://localhost:8888/?token=49d4394bac446e291c6ddaf349c9dbffcd2cdc8c848eb888`, but youts will surely be different!

ATTENTION 2: While Jupyter server is active, you can't put commands in the terminal !

In the console you see the server output of Jupyter, which is active and in certain sense 'it has taken control' of the terminal. This means that if you write some commands inside the terminal, these **will not** be executed!

3.3.2 Saving Jupyter notebooks

You can save the current notebook in Jupyter by pressing `Control-S` while in the browser.

ATTENTION: DO NOT OPEN THE SAME DOCUMENT IN MANY TABS !!

Be careful to not open the same notebook in more the one tab, as modifications in different tabs may overwrite at random ! To avoid these awful situations, make sure to have only one tab per document. If you accidentally open the same notebook in different tabs, just close the additional tab.

Automated savings

Notebook changes are automatically saved every few minutes.

3.3.3 Turning off Jupyter server

Before closing Jupyter server, remember to save in the browser the notebooks you modified so far.

To correctly close Jupyter, *do not* brutally close the terminal, Instead, from the the terminal where you ran Jupyter, hit `Control-c`, a question should appear to which you should answer `y` (if you don't answer in 5 seconds, you will have to hit `control-c` again).

```
Shutdown this notebook server (y/[n])? y
[C 11:05:03.062 NotebookApp] Shutdown confirmed
[I 11:05:03.064 NotebookApp] Shutting down kernels
```

3.3.4 Navigating notebooks

(Optional) To improve navigation experience in Jupyter notebooks, you may want to install some Jupyter extension, like `toc2` which shows paragraph headers in the sidebar. To install:

Install the Jupyter contrib extensions[58]:

1a. If you have Anaconda: Open Anaconda Prompt, and type:

```
conda install -c conda-forge jupyter_contrib_nbextensions
```

1b. If you don't have Anaconda: Open the terminal and type:

```
python3 -m pip install --user jupyter_contrib_nbextensions
```

2. Install in Jupyter:

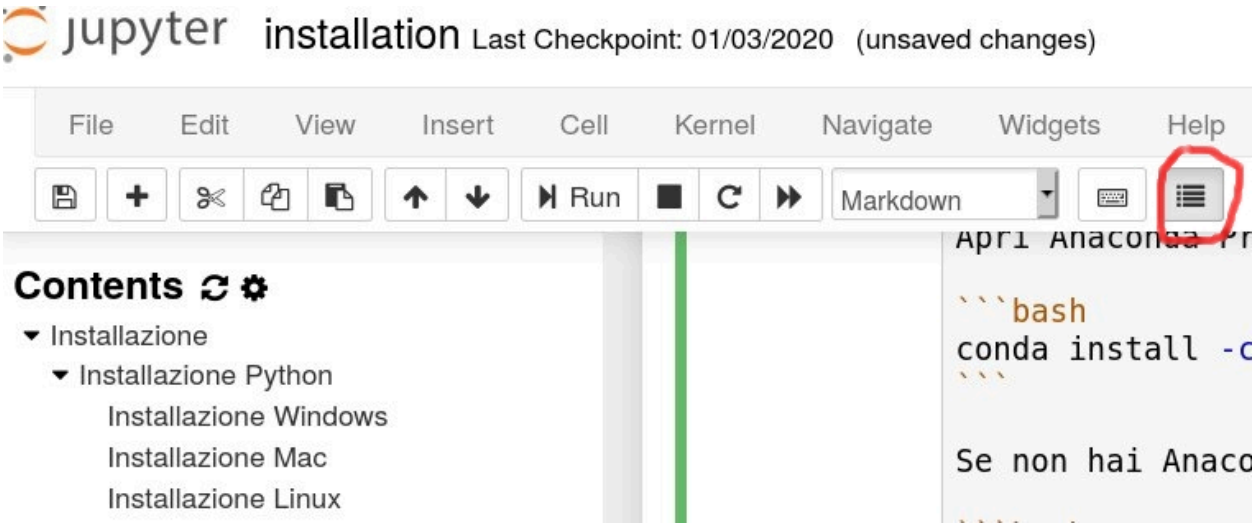
```
jupyter contrib nbextension install --user
```

3. Enable extensions:

```
jupyter nbextension enable toc2/main
```

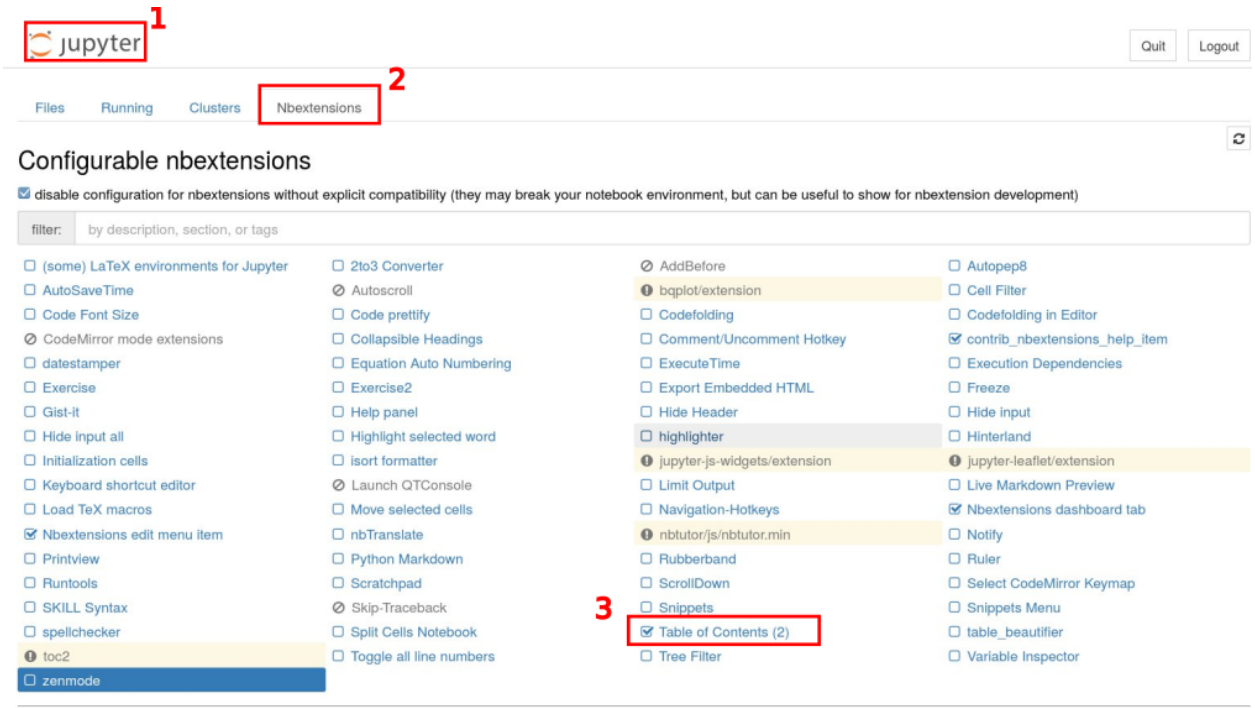
[58] https://github.com/ipython-contrib/jupyter_contrib_nbextensions

Once installed: To see table of contents in a document you will have to press a list button on the right side of the toolbar:



If by chance you don't see the button:

1. go to main Jupyter interface
2. check `Nbextensions` tab
3. make sure `Table of Contents(2)` is enabled
4. Close Jupyter, reopen it, go to a notebook, you should finally see the button



3.3.5 Installing Jupyter notebook - all operating systems

If you didn't manage to *find and/or start Jupyter*, probably it means we need to install it!

You may try installing Jupyter with `pip` (the native package manager of Python)

To install, run this command:

```
python3 -m pip install --user jupyter -U
```

Once installed, follow the section

Una volta installato, segui la sezione *Run Jupyter Notebook*

ATTENTION: you DON'T need to install Jupyter inside *virtual environments* You can consider Jupyter as a system-level application, which should be independent from virtual environments. If you are inside a virtual environment (i.e. the command line begins with a writing in parenthesis like (myprj)``), exit the environment by typeing``deactivate`)

HELP: if you have trouble installing Jupyter, while waiting for help you can always try the online demo version[59] (note: it's not always available) or Google Colab[60]

3.4 Projects with virtual environments

WARNING: If these are your first steps with Python, you can skip this section.

You should read it if you have already done personal projects with Python that you want to avoid compromising, or when you want to make a project to ship to somebody.

When we start a new project with Python, we usually notice quickly that we need to extend Python with particular libraries, like for example to draw charts. Not only that, we might also want to install Python programs which are not written by us and they might as well need their peculiar libraries to work.

Now, we could install all these extra libraries in a unique cauldron for the whole computer, but each project may require its specific versions of each library, and sometimes it might not like versions already installed by other projects. Even worse, it might automatically update packages used by old projects, preventing old code from working anymore. So it is PRACTICALLY NECESSARY to separate well each project and its dependencies from those of other projects: for this purpose you can create a so-called *virtual environment* .

[59] https://try.jupyter.org/
[60] http://colab.research.google.com/

3.4.1 Creating virtual environments

- **If you installed Anaconda**, to create virtual environments you can use its package manager `conda`. Supposing we want to call our project `myprj` (but it could be any name), to put into a folder with the same name `myprj`, we can use this command to create a virtual environment:

```
conda create -n myprj
```

  The command might require you to download packages, you can safely confirm.

- **If you *don't have* Anaconda installed**, to create virtual environments it's best to use the native Python module `venv`:

```
python3 -m venv myprj
```

Both methods create the folder `myprj` and fill it with all required Pyhton files to have a project completely isolated from the rest of the computer. But now, how can we tell Python we want to work right with that project? We must *activate* the environment as follows.

3.4.2 Activate a virtual environment

To activate the virtual environment, we must use different commands according to our operating system (but always from the terminal)

Activate environment in Windows with Anaconda:

```
activate myprj
```

Linux & Mac (without Anaconda):

```
source myprj/bin/activate
```

Once the environment is active, in the command prompt we should see the name of that environment (in this case `myprj`) between round parenthesis at the beginning of the row:

```
(myprj) some/current/folder >
```

The prefix lets us know that the environment `myprj` is currently active, so Python comands we will use all use the settings and libraries of that environment.

Note: inside the virtual environment, we can use the command `python` instead of `python3` and `pip` instead of `pip3`

Deactivate an environment:

Write in the console the command `deactivate`. Once the environment is deactivated, the environment name `(myprj)` at the beginning of the prompt should disappear.

3.4.3 Executing environments inside Jupyter

As we said before, Jupyter is a system-level application, so there should be one and only one Jupyter. Nevertheless, during Jupyter execution, we might want to execute our Python commands in a particular Python environment. To do so, we must configure Jupyter so to use the desired environment. In Jupyter terminology, the configurations are called *kernel*: they define the programs launched by Jupyter (be they Python versions or also other languages like R). The current kernel for a notebook is visibile in the right-upper corner. To select a desired kernel, there are several ways:

With Anaconda

Jupyter should be available in the Navigator. If in the Navigator you enable an environment (like for example Python 3), when you then launch Jupyter and create a notebook you should have the desired environment active, or at least be able to select a kernel with that environment.

Without Anaconda

In this case, the procedure is a little more complex:

— 1 From the terminal þactivate your environment](#Activate-a-virtual-environment)

— 2 Create a Jupyter kernel:

```
python3 -m ipykernel install --user --name myprj
```

NOTE: here `myprj` is the name of the *Jupter kernel.* We use the same name of the environment only for practical reasons.

— 3 Deactivate your environment, by launching

```
deactivate
```

From now on, every time you run Jupyter, if everything went well under the `Kernel` menu in the notebook you should be able to select the kernel just created (in this example, it should have the name `myprj`)

NOTE: the passage to create the kernel must be done only once per project

---

NOTE: you don't need to activate the environment before running Jupyter!

During the execution of Jupyter simply select the desired kernel. Nevertheless, it is convenient to execute Jupyter from the folder of our virtual environment, so we will see all the project files in the Jupyter home.

3.5 Further readings

Go on with the page Tools and scripts[61] to learn how to use other editors and Python architecture.

[61] https://en.softpython.org/tools/tools-sol.html

CHAPTER

FOUR

A - FOUNDATIONS

4.1 Commandments

The Supreme Committee for the Doctrine of Coding has ruled important Commandments you shall follow.

If you accept their wise words, you shall become a true Python Jedi.

> **WARNING**: if you don't follow the Commandments, you will end up in *Debugging Hell* !

4.1.1 I COMANDAMENT

You shall write Python code

Who does not writes Python code, does not learn Python

4.1.2 II COMANDAMENT

Whenever you insert a variable in a `for` cycle, such variables must be new

If you defined the variable before, you shall not reintroduce it in a `for`, because doing so might bring confusion in the minds of the readers.

So avoid such sins:

[1]:
```
i = 7
for i in range(3):  # sin, you lose variable i
    print(i)

print(i)   # prints 2 and not 7  !!
```

```
0
1
2
2
```

[2]:
```
def f(i):
    for i in range(3): # sin, you lose parameter i
        print(i)

    print(i)  # prints 2, not the 7 we passed!

f(7)
```

```
0
1
2
2
```

[3]:
```
for i in range(2):

    for i in  range(5):  # debugging hell, you lose the i of external cycle
        print(i)

    print(i)  # prints 4 !!
```

```
0
1
2
3
4
4
0
1
2
3
4
4
```

4.1.3 III COMANDAMENT

You shall never reassign function parameters

You shall never ever perform any of these assignments, as you risk losing the parameter passed during function call:

[4]:
```
def sin(my_int):
    my_int = 666            # you lost the 5 passed from external call!
    print(my_int)           # prints 666

x = 5
sin(x)
```

```
666
```

Same reasoning can be applied to all other types:

[5]:
```
def evil(my_string):
    my_string = "666"
```

[6]:
```
def disgrace(my_list):
    my_list = [666]
```

[7]:
```
def delirium(my_dict):
    my_dict = {"evil":666}
```

For the sole case when you have composite parameters like lists or dictionaries, you can write like below IF AND ONLY IF the function description requires to MODIFY the internal elements of the parameter (like for example sorting a list in-place or changing the field of a dictionary.

[8]:
```
# MODIFY my_list in some way
def allowed(my_list):
    my_list[2] = 9

outside = [8,5,7]
allowed(outside)
print(outside)
```

```
[8, 5, 9]
```

On the other hand, if the function requires to RETURN a NEW object, you shall not fall into the temptation of modifying the input:

[9]:
```
# RETURN a NEW sorted list
def pain(my_list):
    my_list.sort()    # BAD, you are modifying the input list instead of creating a␣
↪new one!
    return my_list
```

[10]:
```
# RETURN a NEW list
def crisis(my_list):
    my_list[0] = 5           # BAD, as above
    return my_list
```

[11]:
```
# RETURN a NEW dictionary
def tormento(my_dict):
    my_dict['a'] = 6  # BAD, you are modifying the input dictionary instead of␣
↪creating a new one!
    return my_dict
```

[12]:
```
# RETURN a NEW class instance
def desperation(my_instance):
    my_instance.my_field = 6  # BAD, you are modifying the input object
                              #      instead of creating a new one!
    return istanza_di_classe
```

4.1.4 IV COMANDAMENT

You shall never ever reassign values to function calls or mmethods

WRONG:

```
mia_funzione() = 666
mia_funzione() = 'evil'
mia_funzione() = [666]
```

CORRECT:

```
x = 5
y = my_fun()
z = []
z[0] = 7
d = dict()
d["a"] = 6
```

Function calls like `my_function()` return calculations results and store them in a box in memory which is only created for the purposes of the call, and Python will not allow us to reuse it like it were a variabile.

Whenever you see `name()` in the left part, it *cannot* be followed by the equality sign = (but it can be followed by due equals sign == if you are doing a comparison).

4.1.5 V COMMANDMENT

You shall never ever redefine system functions

Python has several system defined functions. For example `list` is a Python type: as such, you can use it for example as a function to convert some type to a list:

```
[13]: list("ciao")
```

```
[13]: ['c', 'i', 'a', 'o']
```

When you allow the Forces of Evil to take the best of you, you might be tempted to use reserved words like `list` as a variable for you own miserable purposes:

```
[14]: list = ['my', 'pitiful', 'list']
```

Python allows you to do so, but we do **not**, for the consequences are disastrous.

For example, if you now attempt to use `list` for its intended purpose like casting to list, it won't work anymore:

```
list("ciao")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-4-c63add832213> in <module>()
----> 1 list("ciao")

TypeError: 'list' object is not callable
```

In particular, we recommend to **not redefine** these precious functions:

- `bool`, `int`,`float`,`tuple`,`str`,`list`,`set`,`dict`
- `max`, `min`, `sum`
- `next`, `iter`
- `id`, `dir`, `vars`,`help`

4.1.6 VI COMMANDMENT

You shall use `return` command only if you see written RETURN in function description!

If there is no `return` in function description, the function is intended to return `None`. In this case you don't even need to write `return None`, as Python will do it implicitly for you.

4.1.7 VII COMMANDMENT

You shall also write on paper!

If staring at the monitor doesn't work, help yourself and draw a representation of the state sof the program. Tables, nodes, arrows, all can help figuring out a solution for the problem.

4.1.8 VIII COMANDAMENT

You shall never ever reassing `self` !

You shall never write horror such as this:

[15]:
```
class MyClass:
    def my_method(self):
        self = {'my_field':666}     # SIN
```

Since `self` is a kind of a dictionary, you might be tempted to write like above, but to external world it will bring no effect.

For example, let's suppose somebody from outside makes a call like this:

[16]:
```
mc = MyClass()
mc.my_method()
```

After the call `mc` will not point to `{'my_field':666}`

[17]:
```
mc
```

[17]:
```
<__main__.MyClass at 0x7f547c35bf28>
```

and will not have `my_field`:

mc.my_field

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-26-5c4e6630908d> in <module>()
----> 1 mc.male

AttributeError: 'MyClass' object has no attribute 'my_field'
```

For the same reasoning, you shall never reassign `self` to lists or others things:

[18]:
```
class MyClass:
    def my_method(self):
        self = ['evil']      # YET ANOTHER SIN
        self = 666           # NO NO NO
```

4.1.9 IX COMMANDMENT

You shall test!

Untested code by definition *does not work*. For ideas on how to test it, have a look at Errors and testing[62]

4.1.10 X COMMANDMENT

You shall never ever add or remove elements from a sequence you are iterating with a `for` !

Falling into such temptations **would produce totally unpredictable behaviours** (do you know the expression *pulling the rug out from under your feet ?*)

Do not add, because you risk to walk on a tapis roulant that never turns off:

```
my_list = ['a','b','c','d','e']
for el in my_list:
    my_list.append(el)  # YOU ARE CLOGGING COMPUTER MEMORY
```

Do not remove, because you risk to corrupt the natural order of things:

[19]:
```
my_list = ['a','b','c','d','e']

for el in my_list:
    my_list.remove(el)    # VERY BAD IDEA
```

Look at the code. You think we removed eveything, uh?

[20]:
```
my_list
```

[20]:
```
['b', 'd']
```

`O_o'` Do not even try to make sense of such sorcery - nobody can, because it is related to internal implmentation of Python.

My version of Python gives this absurd result, yours may give another. Same applies for iteration on sets and dictionaries. **You are warned**.

If you really need to remove stuff from the sequence you are iterating on, use a while cycle[63] or make first a copy of the original sequence.

[62] https://en.softpython.org/errors-and-testing/errors-and-testing-sol.ipynb
[63] https://en.softpython.org/control-flow/flow3-while-sol.html

CHAPTER

FIVE

CHAPTER

SIX

INDEX