Research Track II
Jupyter Notebook

Carmine Tommaso Recchiuto

Jupyter Notebook

- The Jupyter Notebook is an open source web application that you can use to create and share documents that contain live code, equations, visualizations, and text. Jupyter Notebook is maintained by the people at Project Jupyter.

- Project Jupyter is a non-profit, open-source project, born out of the IPython Project in 2014 as it evolved to support interactive data science and scientific computing across all programming languages

- IPython (short for *Interactive Python*) was started in 2001 by Fernando Perez as an enhanced Python interpreter. As well as being a useful interactive interface to Python, IPython also provides a number of useful syntactic additions to the language. The Jupyter notebook is a browser-based graphical interface to the IPython shell, and builds on it a rich set of dynamic display capabilities.

- Jupyter Notebooks are a spin-off project from the IPython project, which used to have an IPython Notebook project itself.

Jupyter Notebook

- The name, Jupyter, comes from the core supported programming languages that it supports: Julia, Python, and R. Jupyter ships with the IPython kernel, which allows you to write your programs in Python, but there are currently over 100 other kernels that you can also use.

- The Jupyter Notebook is not included with Python, so if you want to try it out, you will need to install Jupyter.

pip3 install jupyter bqplot pyyaml ipywidgets
jupyter nbextension enable --py widgetsnbextension

- Now that you have Jupyter installed, let's learn how to use it. To get started, all you need to do is open up your terminal application and go to a folder of your choice. I recommend using something like your Desktop folder to start out with and create a subfolder there called *Notebooks* or something else that is easy to remember. Then run:

jupyter notebook --allow-root (the --allow-root option is only necessary if you are using the Docker Image. Also, in the docker you will probably need to update firefox with ***apt-get install firefox***)

Jupyter and Docker

- ✓ Jupyter may be used in a Docker container (as we are doing in our example) and thanks to the port forwarding option it may be also exposed to the host computer!

However in this case we should start a brand new container, by also forwarding port 8888:

```
docker run -it --name my_jupyter -p 6080:80 -p 5900:5900 -p 8888:8888 carms84/noetic_ros2
```

And you need to start the Jupyter Notebook with:

```
jupyter notebook --allow-root --ip 0.0.0.0
```

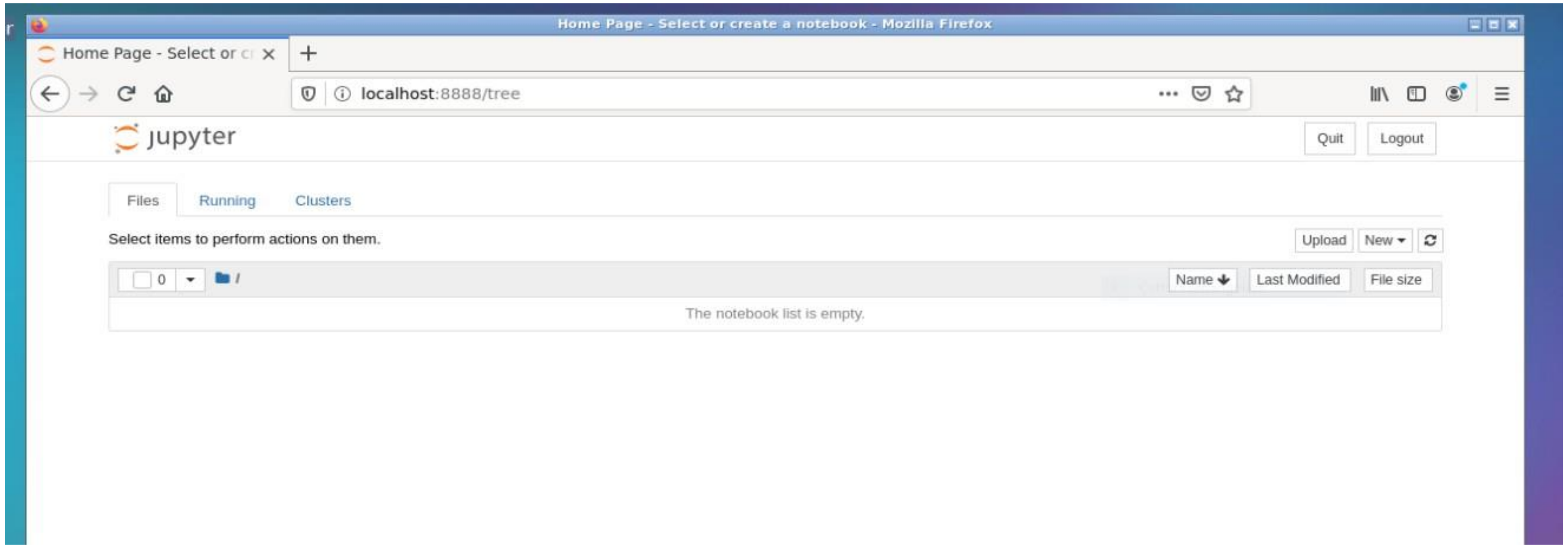
So that it can be visible also outside (on the host side)

Now open a browser on the host and go to:

http://localhost:8888

Use the token specify on the Docker terminal to enter, or to set a password for the next time.

Jupyter Notebook



Select items to perform actions on them.

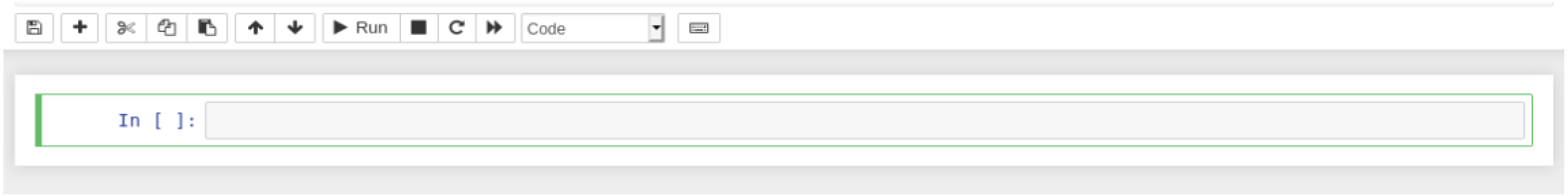
The notebook list is empty.

Note that right now you are not actually running a Notebook, but instead you are just running the Notebook server. Your default browser should start (or open a new tab) to the following URL: http://localhost:8888/tree, communicating with the Jupyter Notebook server which is running on the terminal shell

Let’s actually create a Notebook now!

First Document

Clicking on the *New* button (upper right), it will open up a list of choices, which will depend on the python versions installed in your machine, in the Docker image you should have only Python3 installed. You have now started your first Jupyter document, and your web page should now look like this:



You will notice that at the top of the page is the word *Untitled*. This is the title for the page and the name of your Notebook. Since that isn't a very descriptive name, let's change it!
Just move your mouse over the word *Untitled* and click on the text. You should now see an in-browser dialog titled *Rename Notebook*. Let's rename this one to *Hello Jupyter*.

If you check the tree of your notebook server, you may see that we have now a jupyter document running (extension .ipynb)

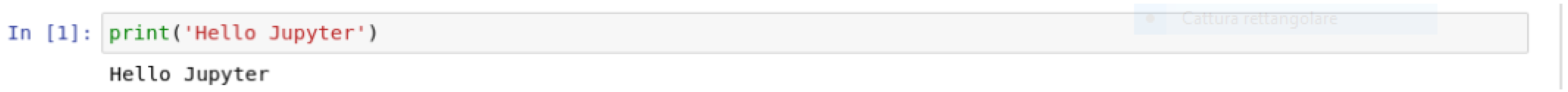
First Document

- ✓ As default, a Notebook’s cell will be used to contain code whenever you first create one, and that cell uses the kernel that you chose when you started your Notebook.
- ✓ In this case, you started yours with Python 3 as your kernel, so that means you can write Python code in your code cells. Since your initial Notebook has only one empty cell in it, the Notebook can’t really do anything.

- ✓ Thus, to verify that everything is working as it should, you can add some Python code to the cell and try running its contents.
- ✓ Let’s try adding the following code to that cell:

print('Hello Jupyter!')

Running a cell means that you will execute the cell’s contents. To execute a cell, you can just select the cell and click the *Run* button that is in the row of buttons along the top. If you prefer using your keyboard, you can just press Shift+Enter



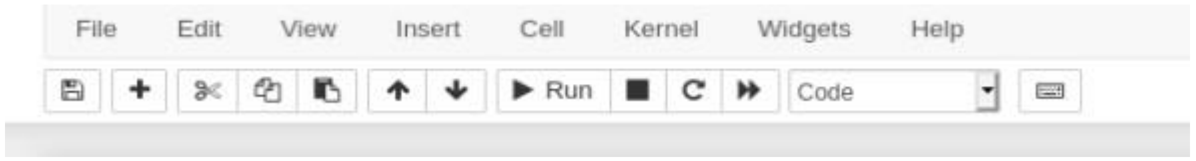
```
In [1]: print('Hello Jupyter')

        Hello Jupyter
```

Jupyter Basics

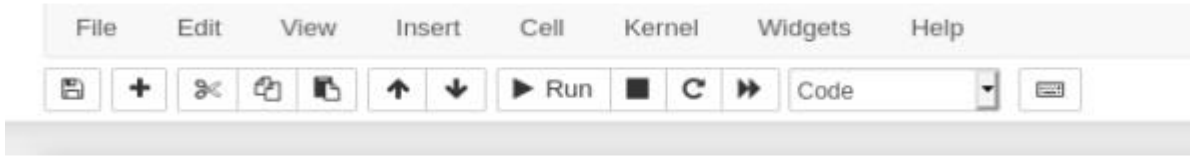
- ✓ If you have multiple cells in your Notebook, and you run the cells in order, you can share your variables and imports across cells. This makes it easy to separate out your code into logical chunks without needing to reimport libraries or recreate variables or functions in every cell.

- ✓ When you run a cell, you will notice that there are some square braces next to the word *In* to the left of the cell. The square braces will auto fill with a number that indicates the order that you ran the cells. For example, if you open a fresh Notebook and run the first cell at the top of the Notebook, the square braces will fill with the number *1*.

- ✓ Let' s now focus a bit on the user interface of the Jupyter Notebook. You may see that there are several menus that you can use to interact with your document.



Jupyter Basics

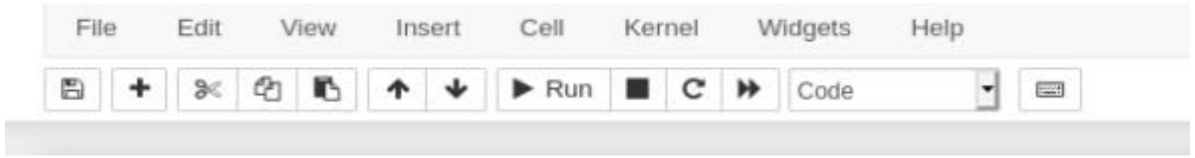


- ✓ The first menu is the File menu. In it, you can create a new Notebook or open a preexisting one. This is also where you would go to rename a Notebook. You have here also the *Save and Checkpoint* option. This allows you to create checkpoints that you can roll back to if you need to.

- ✓ Next is the *Edit* menu. Here you can cut, copy, and paste cells. This is also where you would go if you wanted to delete, split, or merge a cell. You can reorder cells here too. Note that some of the items in this menu are greyed out. The reason for this is that they do not apply to the currently selected cell. For example, a code cell cannot have an image inserted into it, but a Markdown cell can. If you see a greyed out menu item, try changing the cell's type and see if the item becomes available to use.

- ✓ Indeed, the *Cell* menu allows you to run one cell, a group of cells, or all the cells. You can also go here to change a cell's type, although the toolbar may be more intuitive for that. The other handy feature in this menu is the ability to clear a cell's output. If you are planning to share your Notebook with others, you will probably want to clear the output first so that the next person can run the cells themselves.

Jupyter Basics



- ✓ The *View* menu is useful for toggling the visibility of the header and toolbar. You can also toggle *Line Numbers* within cells on or off, or modify the cell's toolbar.
- ✓ The *Insert* menu is just for inserting cells above or below the currently selected cell.
- ✓ The *Kernel* cell is for working with the kernel that is running in the background. Here you can restart the kernel, reconnect to it, shut it down, or even change which kernel your Notebook is using. When debugging a Notebook Document, you may need to restart the Kernel
- ✓ The *Widgets* menu is for saving and clearing widget state. Widgets are basically JavaScript widgets that you can add to your cells to make dynamic content using Python (or another Kernel).
- ✓ Finally you have the Help menu, which is where you go to learn about the Notebook's keyboard shortcuts, a user interface tour, and lots of reference material.

Jupyter Basics

- ✓ Jupyter Notebook also allows you to start more than just Notebooks. You can also create a text file, a folder, or a Terminal in your browser. Go back to the home page that opened when you first started the Jupyter server at http://localhost:8888/tree. Go to the New button and choose one of the other options.
- ✓ The Terminal is quite interesting, as it is like running your operating systems terminal in the browser. This allows you to run any shell command that you might need directly in your browsers.
- ✓ Also on the home page of your Jupyter server (http://localhost:8888/tree) are two other tabs: Running and Clusters.
- ✓ The Running tab will tell you which Notebooks and Terminals you are currently running. This is useful for when you want to shut down your server but you need to make sure that you have saved all your data. However Notebooks auto-save pretty frequently, so you rarely lose data.
- ✓ Clusters may be used for parallel computing: in this case you will need to used ipyparallel on the server.

Jupyter Basics

Jupyter Notebook supports adding rich content to its cells. In this section, you will get an overview of just some of the things you can do with your cells using Markup and Code.

Cell Types

There are technically four cell types: Code, Markdown, Raw NBConvert, and Heading.

The Heading cell type is no longer supported. Instead, you are supposed to use Markdown for your Headings.

The Raw NBConvert cell type is only intended for special use cases when using the nbconvert command line tool.
Basically it allows you to control the formatting in a very specific way when converting from a Notebook to another format.

The primary cell types that you will use are the Code and Markdown cell types.

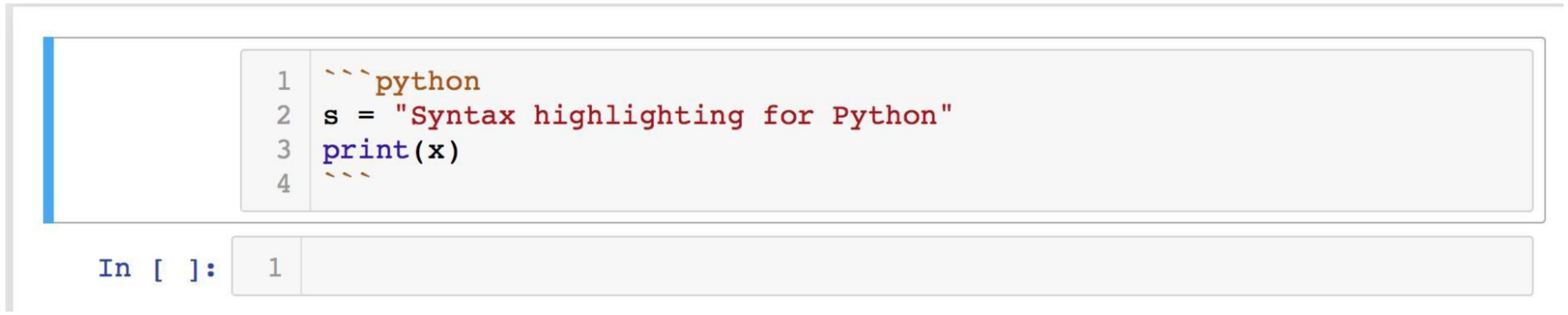
Markdown Cells

Jupyter Notebook supports Markdown, which is a markup language that is a superset of HTML.

So, for example you can create headers by using the pound sign. The more pound signs you use, the smaller the header. Jupyter Notebook even kind of previews it for you.

You can create a list (bullet points) by using dashes, plus signs, or asterisks.

If you want to insert a code example that you don't want your end user to actually run, you can use Markdown to insert it. For inline code highlighting, just surround the code with backticks. If you want to insert a block of code, you can use triple backticks (`) and also specify the programming language:



~~~
```python
s = "Syntax highlighting for Python"
print(x)
```
~~~

Cells Magic

A markdown or code cell may be enriched with some additional functionalities by using some *magic functions:*

%%html renders contents of code cell as html script

Es. *<font size=6 color='blue'>Hello Jupyter</font>*

%%js or *%%javascript* allows for embed javacript code in Jupyter notebook cell:

```
function add(x,y)
{
   var z = x+y;
   alert("x+y="+z.toString());
}
add(10,20)
```

%%writefile writes to a file the contents of code cells

Jupyter Lab

JupyterLab is the next-generation user interface, including notebooks. It has a modular structure, where you can open several notebooks or files (e.g., HTML, Text, Markdowns, etc.) as tabs in the same window.

To install it: **pip3 install jupyterlab**

To start it: **jupyter lab --allow-root --ip 0.0.0.0**

JupyterLab uses the same Notebook server and file format as the classic Jupyter Notebook to be fully compatible with the existing notebooks and kernels. The Classic Notebook and Jupyterlab can run side to side on the same computer. One can easily switch between the two interfaces.

Main difference:

- JupyterLab runs in a single tab, with sub-tabs displayed within that one tab, Jupyter Notebook opens new notebooks in new tabs.

Exporting Jupyter Documents

- Working with Jupyter Notebooks, you may need to share your results with non-technical people. When that happens, you can use the nbconvert tool which comes with Jupyter Notebook to convert or export your Notebook into one of the following formats: HTML, LaTeX, PDF, Markdown, Executable script and others

- The nbconvert tool uses Jinja templates under the covers to convert your Notebook files (.ipynb) into these other formats. Jinja is a template engine that was made for Python. Also note that nbconvert also depends on additional libraries, such as Pandoc and TeX to be able to export to all the formats above (e.g. ***apt-get install pandoc texlive-xetex***). If you don't have one or more of these, some of the export types may not work. For more information, you should check out the documentation.

- The *nbconvert* command does not take very many parameters, which makes learning how to use it easier. Open up a terminal and navigate to the folder that contains the Notebook you wish to convert. The basic conversion command looks like this:

```
$ jupyter nbconvert <input notebook> --to <output format>
```

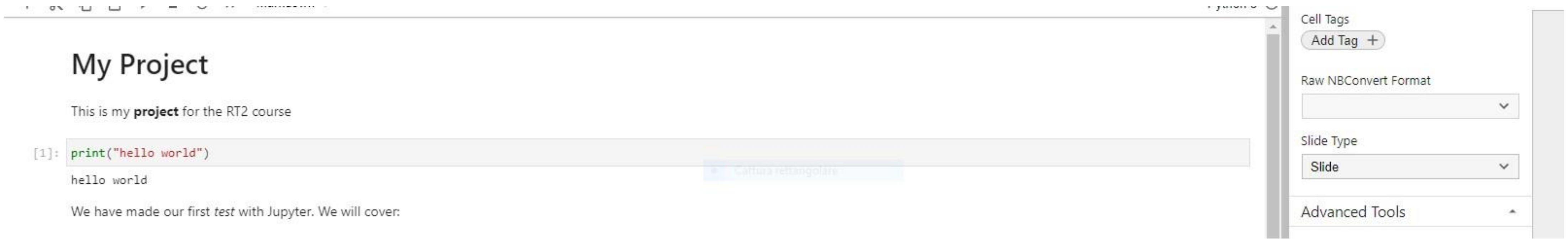
Es. **jupyter nbconvert py_examples.ipynb --to pdf**

Exporting Jupyter Documents

- The conversion process for the other file types is quite similar. You just have to tell nbconvert what type to convert to (PDF, Markdown, HTML, and so on).

- You can also export your currently running Notebook by going to the File menu and choosing the “Export Notebook as” as option. This option allows you to download in all the formats that nbconvert supports. However nbconvert may still need to be used to export multiple Notebooks at once, which is something that the menu does not support.

- Jupyter may also be used to create some fancy presentations. Both with Jupyter Notebooks or Jupyter Lab we can set each cell as a slide, sub-slide or fragment



My Project

This is my **project** for the RT2 course

```
[1]: print("hello world")
hello world
```

We have made our first *test* with Jupyter. We will cover:

Cell Tags
Add Tag +
Raw NBConvert Format
Slide Type: Slide
Advanced Tools

Exporting Jupyter Documents

- With the Jupyter Notebook we can go to View → Cell Toolbar → Slideshow
  After clicking slideshow, a light gray bar will appear above each cell with a scroll down window on the top right.
- At this point, you can select what type of slide each cell should be: a regular slide, or a sub-slide if it is a continuation of the previous slide. You can also skip slides or make them be notes, which is helpful when you've got lots of code that you don't necessarily need for your presentation.
- In order to open this notebook as a slideshow you need to run a command in Terminal:

jupyter nbconvert Slides.ipynb --to slides --post serve

Or you can use the File menu of the Jupyter Notebooks or Lab.

- Jupyter notebook slides offer a simple, clear layout and are incredibly easy to create. While they do not offer the amount of formatting and design features as other presentation applications, they do a very good job of presenting code and data visualizations for technical audiences.

Jupyter Extensions

- While Jupyter Notebooks have lots of functionality built in, you can add new functionality through extensions. A Notebook extension (nbextension) is a JavaScript module that you load in most of the views in the Notebook's frontend. If you are handy with JavaScript, you can even write your own extension

- An extension contains one or more plugins that extend JupyterLab. There are two types of JupyterLab extensions: a *source extension* (which requires a rebuild of JupyterLab when installed), and a *prebuilt extension* (which does not require a rebuild of JupyterLab). Rebuilding JupyterLab requires Node.js to be installed.

- JupyterLab extensions can be installed in a number of ways, including:

  - Python pip or conda packages can include either a source extension or a prebuilt extension. These packages may also include a server-side component necessary for the extension to function.

  - The Extension Manager in JupyterLab and the ***jupyter labextension install*** command can install source extension packages from npm. Installing a source extension requires Node.js and a JupyterLab rebuild to activate

Jupyter Extensions

- To install some of the most common extensions we can:

  - For Jupyter Notebook, run:

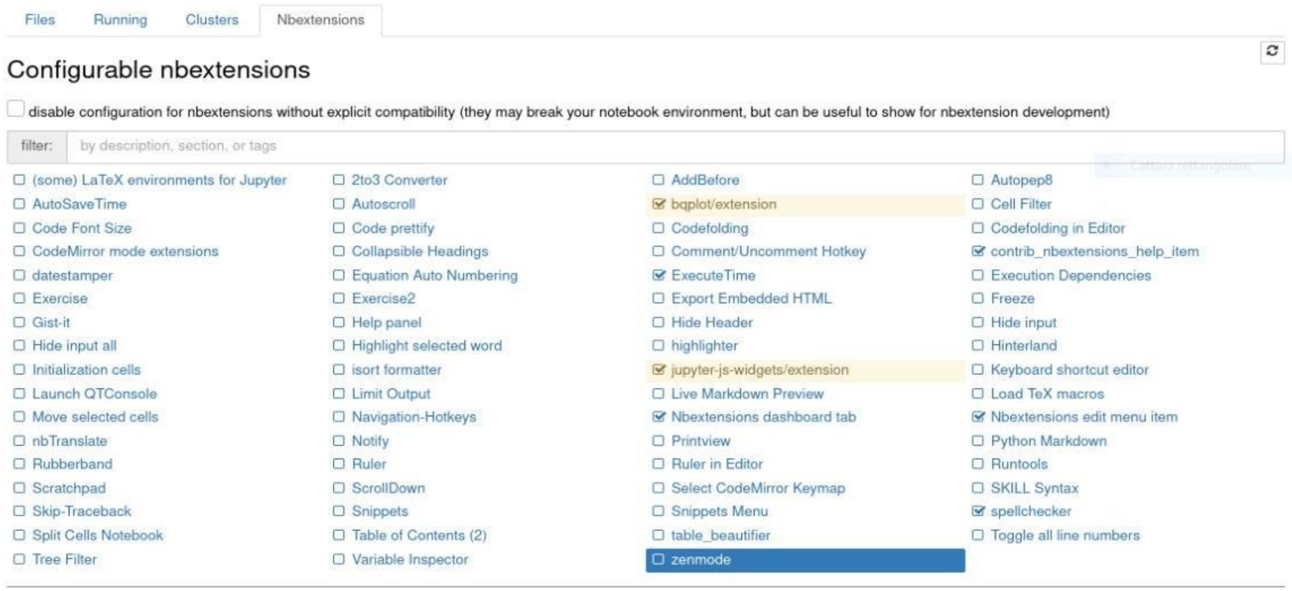
```
pip3 install jupyter_contrib_nbextensions
pip3 install jupyter_nbextensions_configurator
jupyter contrib nbextension install
```

  - For Jupyter Lab, we need to install Node.js > 12.1

```
curl -sL https://deb.nodesource.com/setup_14.x -o nodesource_setup.sh
bash nodesource_setup.sh
apt-get install -y nodejs
```

Jupyter Extensions

- In Jupyter Notebook you can see a new Tab with a list of extensions that can be used to improve the visualization and the usage of the Notebook (untick the box «disable configuration…» in order to select them)



Configurable nbextensions

☐ disable configuration for nbextensions without explicit compatibility (they may break your notebook environment, but can be useful to show for nbextension development)

filter: by description, section, or tags

☐ (some) LaTeX environments for Jupyter	☐ 2to3 Converter	☐ AddBefore	☐ Autopep8
☐ AutoSaveTime	☐ Autoscroll	✓ bqplot/extension	☐ Cell Filter
☐ Code Font Size	☐ Code prettify	☐ Codefolding	☐ Codefolding in Editor
☐ CodeMirror mode extensions	☐ Collapsible Headings	☐ Comment/Uncomment Hotkey	✓ contrib_nbextensions_help_item
☐ datestamper	☐ Equation Auto Numbering	✓ ExecuteTime	☐ Execution Dependencies
☐ Exercise	☐ Exercise2	☐ Export Embedded HTML	☐ Freeze
☐ Gist-it	☐ Help panel	☐ Hide Header	☐ Hide input
☐ Hide input all	☐ Highlight selected word	☐ highlighter	☐ Hinterland
☐ Initialization cells	☐ isort formatter	✓ jupyter-js-widgets/extension	☐ Keyboard shortcut editor
☐ Launch QTConsole	☐ Limit Output	☐ Live Markdown Preview	☐ Load TeX macros
☐ Move selected cells	☐ Navigation-Hotkeys	✓ Nbextensions dashboard tab	✓ Nbextensions edit menu item
☐ nbTranslate	☐ Notify	☐ Printview	☐ Python Markdown
☐ Rubberband	☐ Ruler	☐ Ruler in Editor	☐ Runtools
☐ Scratchpad	☐ ScrollDown	☐ Select CodeMirror Keymap	☐ SKILL Syntax
☐ Skip-Traceback	☐ Snippets	☐ Snippets Menu	✓ spellchecker
☐ Split Cells Notebook	☐ Table of Contents (2)	☐ table_beautifier	☐ Toggle all line numbers
☐ Tree Filter	☐ Variable Inspector	☐ zenmode	

- As an example, let's try to select some of them: spellchecker, tableOfContents, collapsibleHeadings, autopep8, executeTime, lineNumbers, variableInspector, hideCode, skipTraceback,

- For autopep8 you also need to install an additional python module (**pip3 install autopep8**)

Jupyter Extensions

- The spellchecker is an extension for the markdown code. It will just highlight the words that are misspelled.

- The table of contents adds a table of content to the notebook. It can be activated and deactivated with a new button in the menu

- Collapsible headings will let you collapse or expand sections below the headings with toggles that appear on the right-hand sides next to the heading text

- Autopep8 is an extension that helps you format your code according to PEP 8 standards. It will remove empty spaces that are not needed, or add empty lines if required, or apply any other formatting as outlined in PEP8. You can use it with the new “hammer” button in the menu

- ExecuteTime allows you to time how long it took to run a cell and informs you about the time of the latest execution. The information is displayed below each cell.

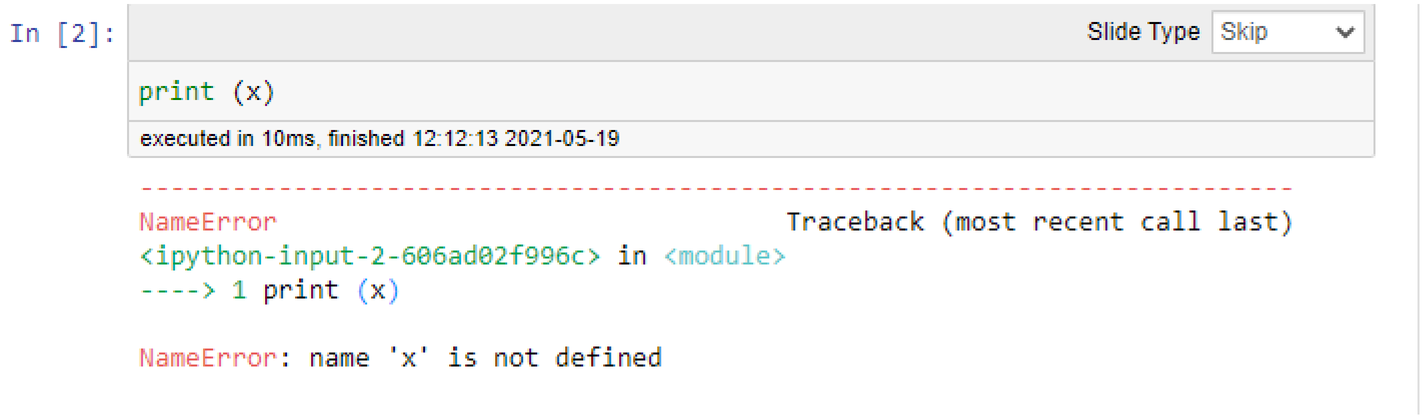
- Toggle all line numbers is a very simple extension that allows you to turn on and off line numbers in code cells using a button in the panel menu.

Jupyter Extensions

- The Variable Inspector is a very useful tool. This extension once enabled can be used by selecting a button marked in red in the panel menu (on the screenshot below).
  Once you select it, you will be shown the information about all variables that you have in the current namespace. You will be able to see variable names, its types, sizes, shapes, and values.

- Hide code allows you to hide all the code in the notebook so you can focus only on the output

- Skip-traceback allows you to skip the traceback when an error in the code is thrown. Instead of the whole traceback, it shows you the error name and short error description.



```
In [2]:                                                    Slide Type  Skip
print (x)
executed in 10ms, finished 12:12:13 2021-05-19

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-606ad02f996c> in <module>
----> 1 print (x)

NameError: name 'x' is not defined
```

Jupyter Extensions

- With Jupyterlab some of the extensions may be directly installed using the Extension Manager (the “puzzle” button)

- However it may happen that some of the extensions are not compatible with the Jupyterlab version

- As an example we can install the spellchecker extension (it’s a source extension package, so we will need to rebuild the kernel) but we cannot install the ExecuteTime extension by using the Extension Manager.

- To this aim, we could still use the terminal and run:

```
pip3 install jupyterlab_execute_time
```

Once restarted the Jupyter server, you should be able to visualize the cell timing.

Some example - Matplotlib

- With Jupyter we can use all the python modules that we can import from our “standard” python shell. As an example, we can use matplotlib.

```
import matplotlib as mpl
import matplotlib.pyplot as plt
```

The IPython kernel of Jupyter notebook is able to display plots of code in input cells. It works seamlessly with **matplotlib** library. The inline option with the **%matplotlib** magic function renders the plot out cell even if **show()** function of plot object is not called.

```
import numpy as np
import math
%matplotlib inline
x = np.arange(0, math.pi*2, 0.05)
y = np.sin(x)
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.plot(x,y)
ax.set_title("sine_wave")
ax.set_xlabel("angle")
ax.set_ylabel("sin")
```

Some example - Matplotlib

- The inline option however only shows the image, without having the possibility of interacting with it. We can use the *widget* option to have an interactive plot. However, please notice that you need to install the matplotlib widget, if you are using the JupiterLab environment (you can do it with the Extension Manager).

- You can notice that by using the ***%matplotlib widget*** option and ***ax.plot(x,y)*** if you modify the data underneath the plot, the display changes dynamically without drawing another plot.

- In Jupyter Notebook, you need to manually install the matplotlib extension with:

  pip3 install ipympl

- You can find a nice example for this widget here:

git clone https://github.com/matplotlib/ipympl.git

We can see from the examples that matplotlib plots can be integrated also with other widgets!

Exercise

- Given the simple turtlesim example:
    - Create a notebook that sets an angular and linear velocity for the turtle
    - Try to implement the same controller implemented in the RT1 course
    - Plots the position of the turtle*

*You can try to implement two different modalities to plot the position of the turtle. You can plot it «live» (more complex) or you can record the position for a certain number of seconds, and then plot the resulting graph.