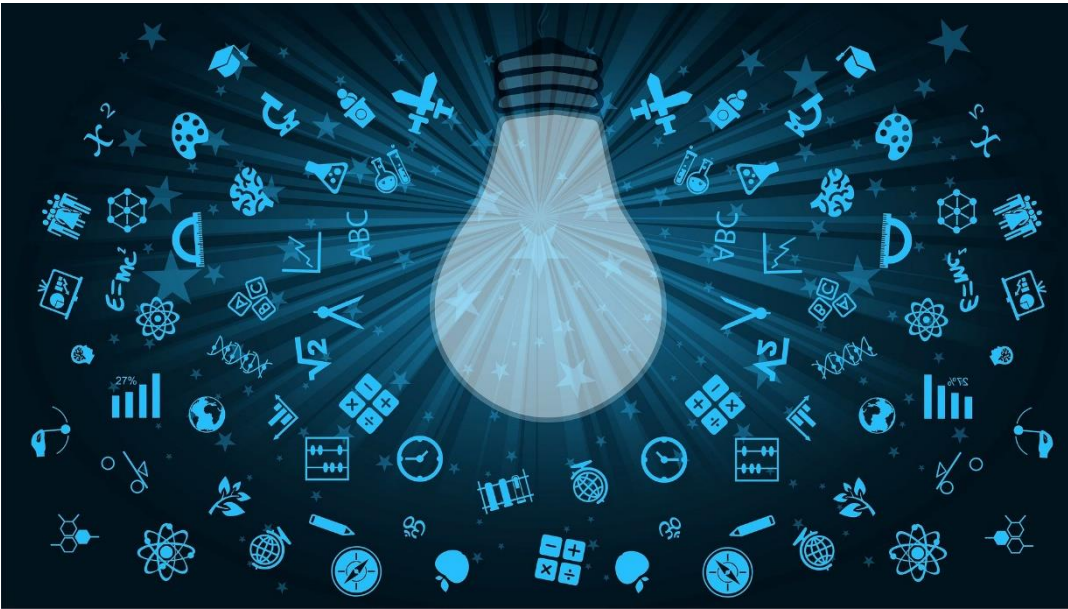




IFPU - Trieste
January 2021

Jupyter Notebook for beginners



The Jupyter Notebook is an incredibly powerful tool for interactively developing and presenting data science projects. A notebook integrates code and its output into a single document that combines visualizations, narrative text, mathematical equations, and other rich media.

The intuitive workflow promotes iterative and rapid development, making notebooks an increasingly popular choice at the heart of contemporary data science, analysis, and increasingly science at large. Best of all, as part of the open source [Project Jupyter](https://projectjupyter.org/), they are completely free.

The Jupyter project is the successor to the earlier IPython Notebook, which was first published as a prototype in 2010. Although it is possible to use many different programming languages within Jupyter Notebooks, this lesson will focus on Python as it is the most common use case.

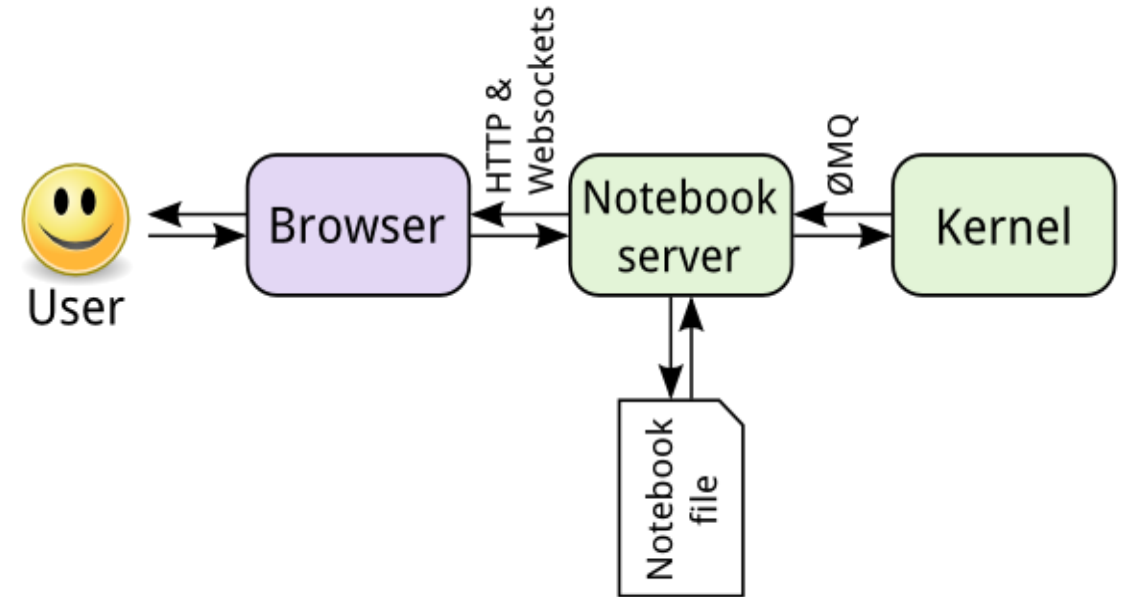
Jupyter Notebooks really shine when you are still in the prototyping phase. This is because **your code is written in independent cells, which are executed individually.**

This allows the user to test a specific block of code in a project without having to execute the code from the start of the script.

What is Jupyter Notebook?

A notebook combines the functionality of:

- a word processor — handles formatted text
- a "*shell*" or "*kernel*" — executes code instructions in a programming language and includes output inline
- a rendering engine — renders HTML in addition to plain text



It is a single document that combines explanations with executable code and its output — an ideal way to provide:

- reproducible research results
- documentation of processes
- instructions
- tutorials and training materials of all shapes and sizes
- A digital learning environment for computational thinking
- **An interactive way to deal with time consuming tasks**

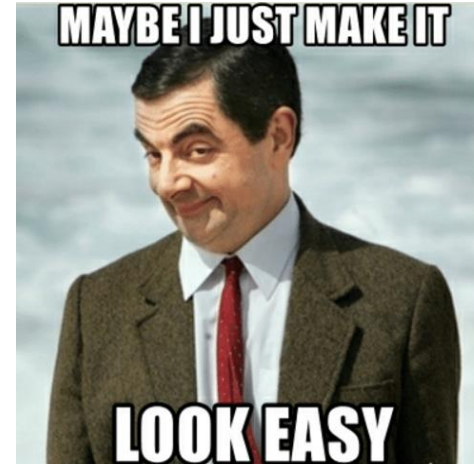
for example, if training your network takes few hours, it is not practical to wait this time to change something in a plot

Installing and start Jupyter Notebook

actually is quite easy:

>> pip install jupyter

that's it!



From your command line enter the command (eventually go at any specific directory path before):

>> jupyter notebook

The current working directory will be the start-up directory (you can only navigate through sub-directories).

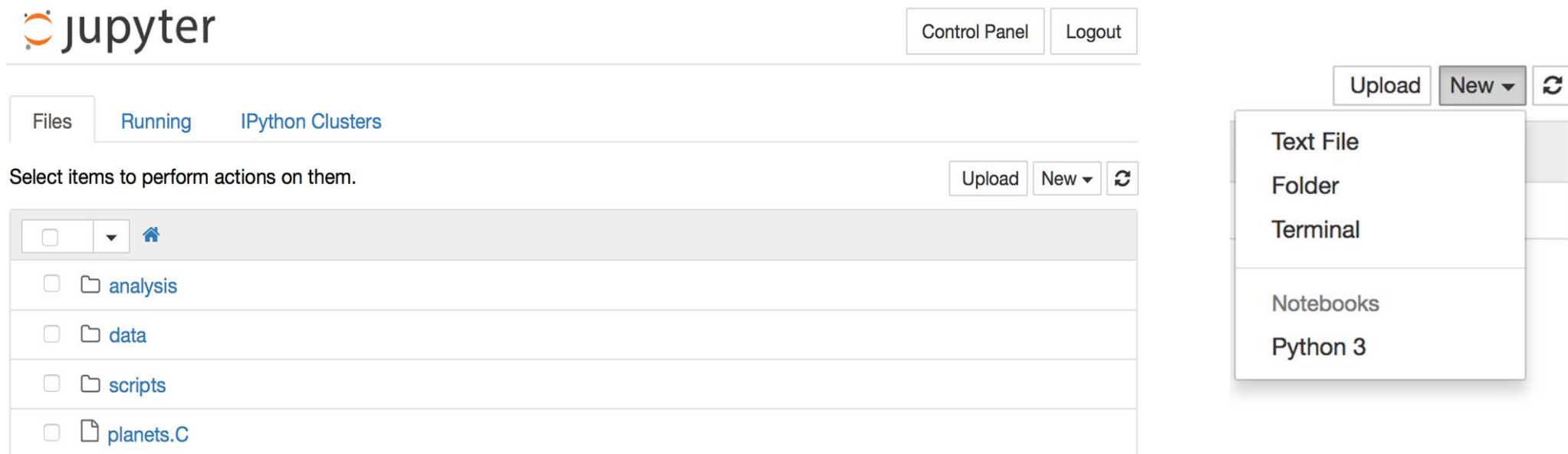
This, depending on your configuration, will start the notebook server and open a page in your browser; something like:
<http://localhost:8888/tree>

it will prompt also an address with a token that could be used to access the specific notebook from any browser (even a remote browser if the port is open), something like:

<http://localhost:8888/?token=666a4f7a78d05058e433bdf63847d5f77bed21637d5e85ca>

Jupyter Notebook dashboard interface

The dashboard's interface is mostly self-explanatory. So far, browse to the folder in which you would like to create your first notebook, click the “New” drop-down button in the top-right and select “Python 3” (or the version of your choice).



here we are! Your first Jupyter Notebook will open in new tab — each notebook uses its own tab because you can open multiple notebooks simultaneously. If you switch back to the dashboard, you will see the new file Untitled.ipynb and you should see some green text that tells you your notebook is running.

What is a Jupyter Notebook file (.ipynb)

It is useful to understand what this file really is.

Each .ipynb file is a text file that describes the contents of your notebook in a format called JSON.

Edit Notebook Metadata



Manually edit the JSON below to manipulate the metadata for this notebook. We recommend putting custom metadata attributes in an appropriately named substructure, so they don't conflict with those of others.

```
1 {
2   "kernelspec": {
3     "name": "python3",
4     "display_name": "Python 3",
5     "language": "python"
6   },
7   "language_info": {
8     "name": "python",
9     "version": "3.7.2",
10    "mimetype": "text/x-python",
11    "codemirror_mode": {
12      "name": "ipython",
13      "version": 3
14    },
15    "pygments_lexer": "ipython3",
16    "nbconvert_exporter": "python",
17    "file_extension": ".py"
18  }
19 }
```

Each cell and its contents, including image attachments that have been converted into strings of text, is listed therein along with some metadata.

You can edit this by yourself — if you know what you are doing! — by selecting:

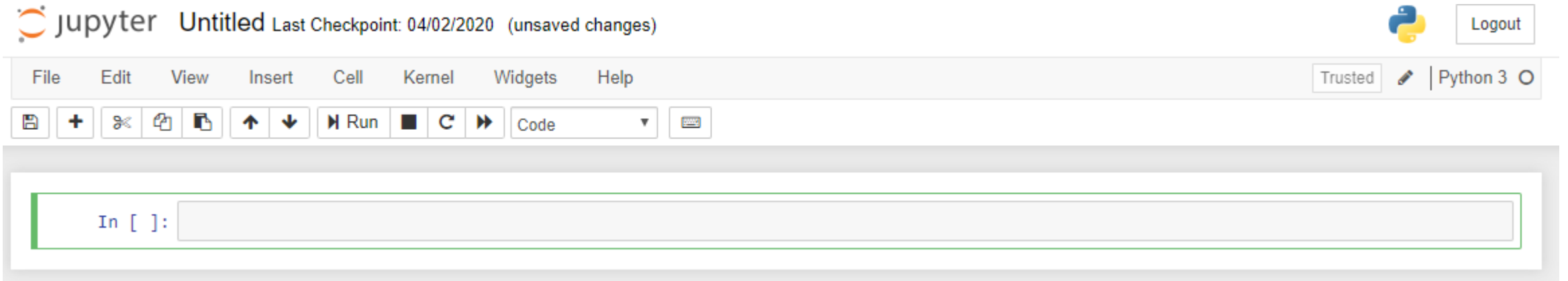
Edit > Edit Notebook Metadata from the menu bar in the notebook.

You can also view the contents of your notebook files by selecting “Edit” from the controls on the dashboard, but the keyword here is “can”; there’s no reason, other than curiosity, to do so, unless you really know what you are doing.

Cancel

Edit

Jupyter Notebook command interface



There are two fairly prominent terms that you should notice, which are probably new to you: cells and kernels are key both to understanding Jupyter and to what makes it more than just a word processor.

Fortunately, these concepts are not difficult to understand.

- A **kernel** is a “computational engine” that executes the code contained in a notebook document.
- A **cell** is a container for text to be displayed in the notebook or code to be executed by the notebook’s kernel.

Jupyter Notebook - Cells

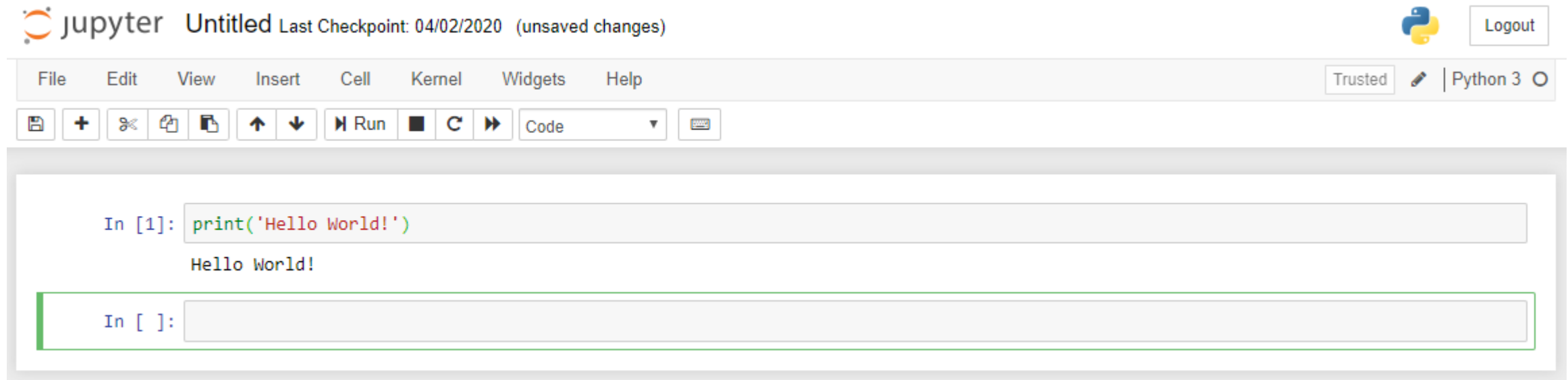
There are two main cell types that we will cover:

- A **code cell** contains code to be executed in the kernel and displays its output below.
- A **Markdown cell** contains text formatted using Markdown and displays its output in-place when it is run.

The first cell in a new notebook is always a code cell.

Let's test it out with a classic hello world example.

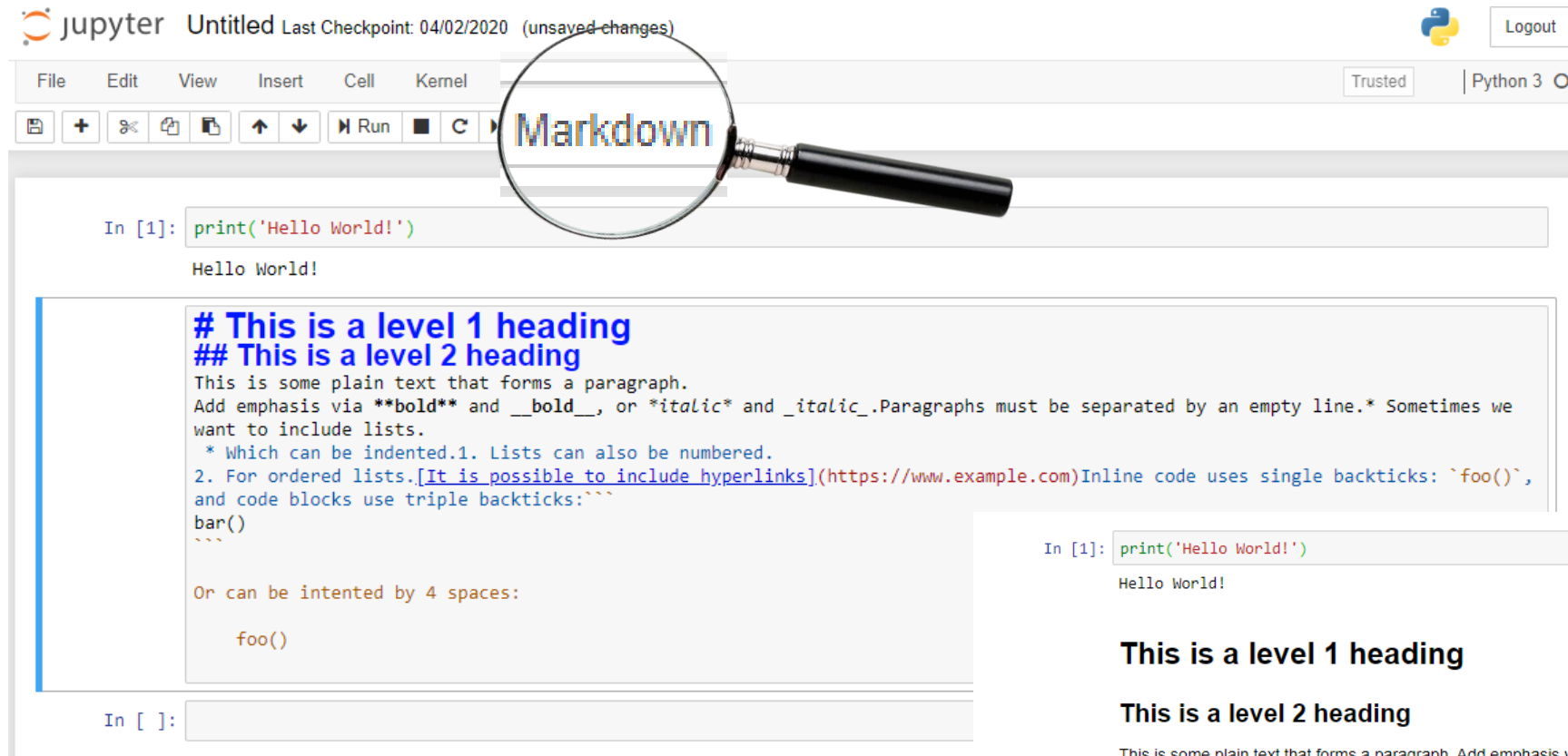
Type ***print('Hello World!')*** into the cell and click the run button in the toolbar above or press Ctrl + Enter.



Jupyter Notebook - Cells

Markdown is a lightweight, easy to learn, markup language for formatting plain text.

Its syntax has a one-to-one correspondence with HTML tags, so some prior knowledge here would be helpful, but is definitely not a prerequisite.



Jupyter Notebook interface showing a code cell and a markdown cell. A magnifying glass highlights the word "Markdown" in the menu bar.

Code cell output:

```
In [1]: print('Hello World!')
```

Hello World!

Markdown cell content:

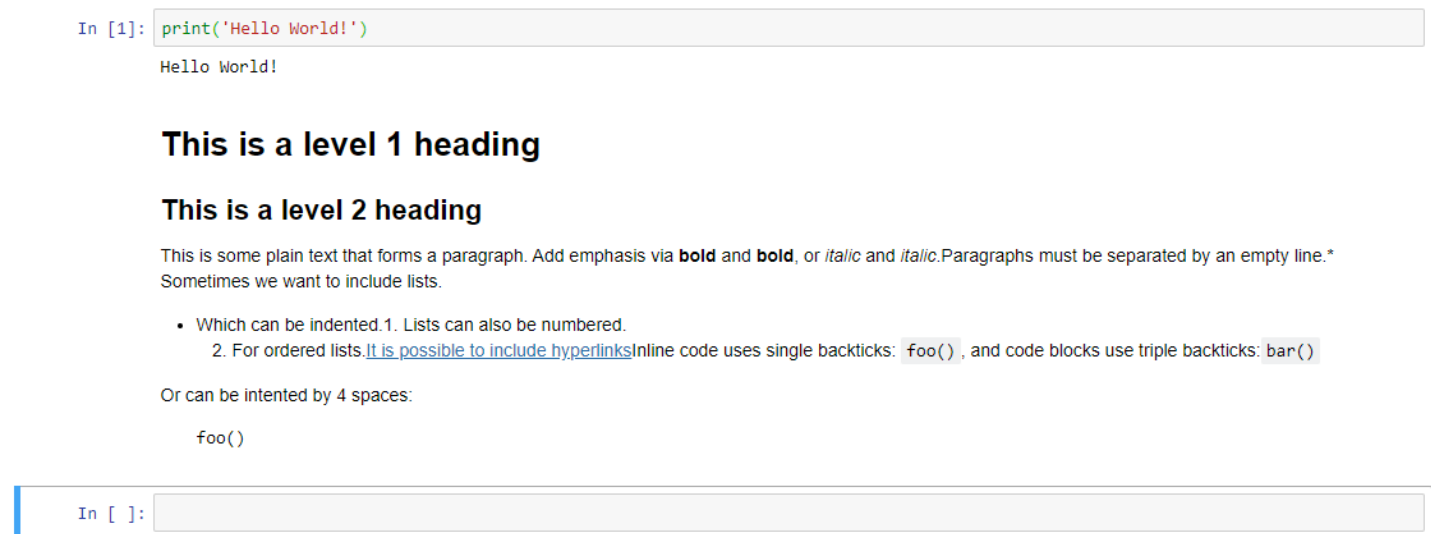
```
# This is a level 1 heading
## This is a level 2 heading
This is some plain text that forms a paragraph.
Add emphasis via bold and bold, or italic and italic. Paragraphs must be separated by an empty line.* Sometimes we
want to include lists.
* Which can be indented.1. Lists can also be numbered.
2. For ordered lists.[It is possible to include hyperlinks](https://www.example.com)Inline code uses single backticks: `foo()`,
and code blocks use triple backticks:```
bar()
```

Or can be indented by 4 spaces:

 foo()
```

Editing the Markdown cell

The Markdown cell execution



Jupyter Notebook interface showing the rendered output of the markdown cell.

Code cell output:

```
In [1]: print('Hello World!')
```

Hello World!

Rendered Markdown cell content:

## This is a level 1 heading

### This is a level 2 heading

This is some plain text that forms a paragraph. Add emphasis via **bold** and bold, or *italic* and italic. Paragraphs must be separated by an empty line.\* Sometimes we want to include lists.

- Which can be indented.1. Lists can also be numbered.
- 2. For ordered lists.[It is possible to include hyperlinks](https://www.example.com)Inline code uses single backticks: `foo()`, and code blocks use triple backticks: 

```
bar()
```

Or can be indented by 4 spaces:

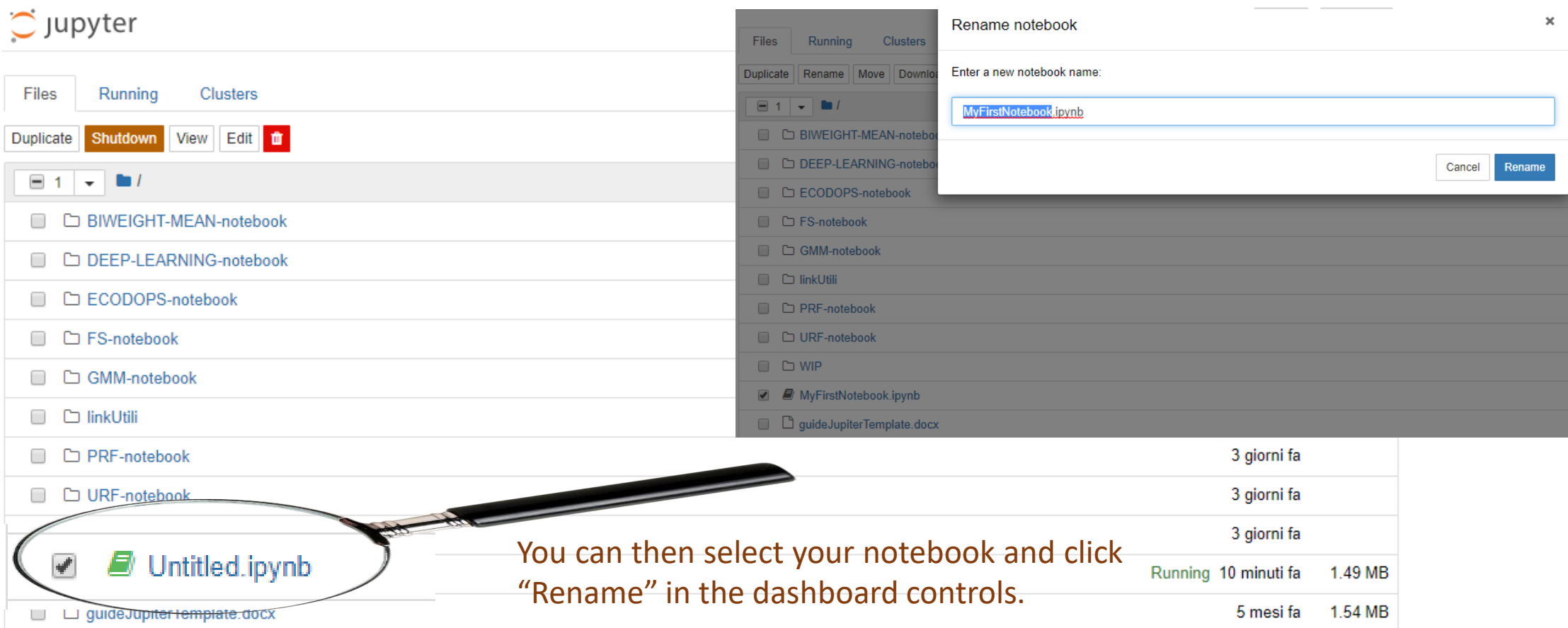
```
 foo()
```

# Jupyter Notebook – naming your notebook

Perhaps somewhat confusingly, you cannot name or rename your notebooks from the notebook app itself, but must use either the dashboard or your file browser to rename the .ipynb file.

You cannot rename a notebook while it is running, so you've first got to shut it down. The easiest way to do this is to select “File > Close and Halt” from the notebook menu.

However, you can also shutdown the kernel either by going to “Kernel > Shutdown” from within the notebook app or by selecting the notebook in the dashboard and clicking “Shutdown” (see image below).



The screenshot shows the Jupyter dashboard interface. At the top, there are tabs for 'Files', 'Running', and 'Clusters'. Below these are buttons for 'Duplicate', 'Shutdown', 'View', 'Edit', and a trash icon. A list of notebooks is displayed, including 'BIWEIGHT-MEAN-notebook', 'DEEP-LEARNING-notebook', 'ECODOPS-notebook', 'FS-notebook', 'GMM-notebook', 'linkUtili', 'PRF-notebook', 'URF-notebook', and 'Untitled.ipynb'. A magnifying glass is positioned over 'Untitled.ipynb'. An inset window titled 'Rename notebook' is shown, with a text input field containing 'MyFirstNotebook.ipynb' and buttons for 'Cancel' and 'Rename'.

| Notebook Name             | Status  | Created      | Size    |
|---------------------------|---------|--------------|---------|
| BIWEIGHT-MEAN-notebook    |         | 3 giorni fa  |         |
| DEEP-LEARNING-notebook    |         | 3 giorni fa  |         |
| ECODOPS-notebook          |         | 3 giorni fa  |         |
| FS-notebook               |         |              |         |
| GMM-notebook              |         |              |         |
| linkUtili                 |         |              |         |
| PRF-notebook              |         |              |         |
| URF-notebook              |         |              |         |
| WIP                       |         |              |         |
| MyFirstNotebook.ipynb     | Running | 10 minuti fa | 1.49 MB |
| guideJupyterTemplate.docx |         | 5 mesi fa    | 1.54 MB |

You can then select your notebook and click “Rename” in the dashboard controls.

# Jupyter Notebook – save & checkpoint

Now we've got started, it's best practice to save regularly.

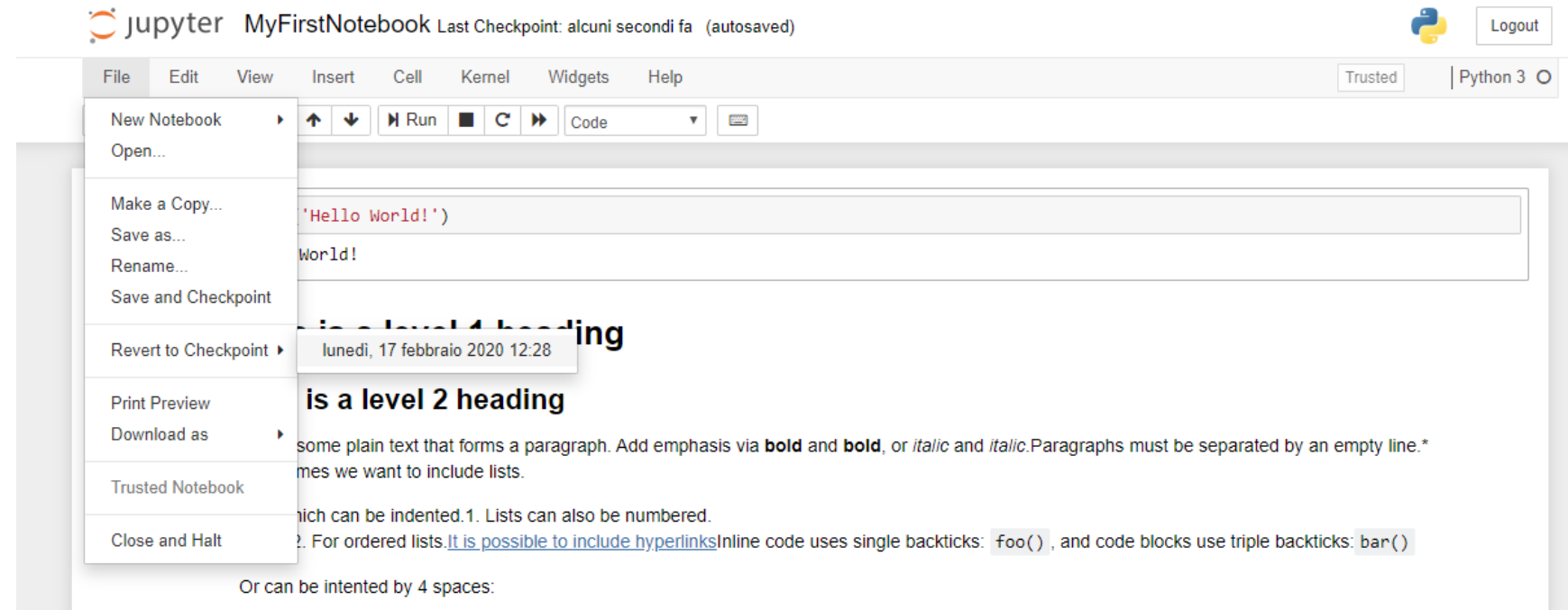
Pressing Ctrl + S will save your notebook by calling the "Save and Checkpoint" command, but what is this checkpoint thing?

Every time you create a new notebook, a checkpoint file is created as well as your notebook file; it will be located within a hidden subdirectory of your save location called `.ipynb_checkpoints` and is also a `.ipynb` file.

By default, Jupyter will auto-save your notebook every 120 seconds to this checkpoint file without altering your primary notebook file.

When you "Save and Checkpoint," both the notebook and checkpoint files are updated. Hence, the checkpoint enables you to recover your unsaved work in the event of an unexpected issue.

You can revert to the checkpoint from the menu via "File > Revert to Checkpoint."



# Jupyter Notebook – exporting your project

Jupyter has built-in support for exporting to HTML and PDF as well as several other formats, which you can find from the menu under “File > Download As.”

If you wish to share your notebooks with a small private group, this functionality may well be all you need.

Indeed, as many researchers in academic institutions are given some public or internal webspace, and because you can export a notebook to an HTML file, Jupyter Notebooks can be an especially convenient way for them to share their results with their peers.

jupyter MyFirstNotebook Last Checkpoint: 5 minuti fa (autosaved)

File Edit View Insert Cell Kernel Widgets Help

New Notebook  
Open...  
Make a Copy...  
Save as...  
Rename...  
Save and Checkpoint

Revert to Checkpoint

Print Preview

Download as

Trusted Notebook

Close and Halt

'Hello World!')

World!

s is a level 1 heading

is a level 2 heading

graph. Add emphasis via **bold** and **bold**, or *italic* and *italic*. Paragraphs must be separated by an

also be numbered.

include [hyperlinks](#)Inline code uses single backticks: `foo()` code blocks use triple backticks

Or can

fo

reST (.rst)

Python (.py)

Reveal.js slides (.slides.html)

```
new 1 x MyFirstNotebook.py x
1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[1]:
5
6
7 print('Hello World!')
8
9
10 # # This is a level 1 heading
11 # ## This is a level 2 heading
12 # This is some plain text that forms a paragraph.
13 # Add emphasis via bold and bold, or italic and italic.
14 # Paragraphs must be separated by an empty line.*
15 # Sometimes we want to include lists.
16 # * Which can be indented.1. Lists can also be numbered.
17 # 2. For ordered lists.[It is possible to include hyperlinks]
18 # (https://www.example.com)Inline code uses single
19 # backticks: `foo()`, and code blocks use triple backticks:``
20 # bar()
21 # ``
22 #
23 # Or can be indented by 4 spaces:
24 #
25 # foo()
26 #
27
28 # ![GraphicsPython.png] (attachment:GraphicsPython.png)
29
30 # In[]:
```

# Jupyter Notebook – basic magics



Magics are handy commands built into the IPython kernel that make it easier to perform particular tasks. Although they often resemble Unix commands, under the hood they are all implemented in Python.

There exist far more magics than it would make sense to cover here, but it's worth highlighting a variety of examples. We will start with a few basics before moving on to more interesting cases.

There are two categories of magic: line magics and cell magics. Respectively, they act on a single line or can be spread across multiple lines or entire cells. Below the available magics:

## Available **line** magics:

%alias %alias\_magic %autocall %automagic %autosave %bookmark %cd %clear %cls %colors %config %connect\_info %copy %ddir %debug %dhist %dirs %doctest\_mode %echo %ed %edit %env %gui %hist %history %killbgscripts %ldir %less %load %load\_ext %loadpy %logoff %login %logstart %logstate %logstop %ls %lsmagic %macro %magic %matplotlib %mkdir %more %notebook %page %pastebin %pdb %pdef %pdoc %pfile %pinfo %pinfo2 %popd %pprint %precision %profile %prun %psearch %psource %pushd %pwd %pycat %pylab %qtconsole %quickref %recall %rehashx %reload\_ext %ren %rep %rerun %reset %reset\_selective %rmdir %run %save %sc %set\_env %store %sx %system %tb %time %timeit %unalias %unload\_ext %who %who\_ls %whos %xdel %xmode

## Available **cell** magics:

%%! %%HTML %%SVG %%bash %%capture %%cmd %%debug %%file %%html %%javascript %%js %%latex %%markdown %%perl %%prun %%pypy %%python %%python2 %%python3 %%ruby %%script %%sh %%svg %%sx %%system %%time %%timeit %%writefile

```
In [10]: %time
```

```
Wall time: 0 ns
```

```
In [12]: %echo ciao
```

```
ciao
```

# Jupyter Notebook - logging

Jupyter has a built-in way to prominently display custom error messages above cell output.

This can be handy for ensuring that errors and warnings about things like invalid inputs or parametrizations are hard to miss for anyone who might be using your notebooks.

An easy, customizable way to hook into this is via the standard Python logging module.

```
In [13]: import logging
logging.error('Jupyter notebook automatic logging')
```

```
ERROR:root:Jupyter notebook automatic logging
```

```
In [15]: logger = logging.getLogger()
logger.setLevel(logging.DEBUG)

logging.info('This is some information')
logging.debug('This is a debug message')
```

```
INFO:root:This is some information
DEBUG:root:This is a debug message
```

# Jupyter Notebook – executing external code



Although the foremost power of Jupyter Notebooks emanates from their interactive flow, it is also possible to run notebooks in a non-interactive mode. Executing notebooks from scripts or the command line provides a powerful way to produce automated reports or similar documents.

Jupyter offers a command line tool that can be used, in its simplest form, for file conversion and execution. As you are probably aware, notebooks can be converted to a number of formats, available from the UI under “File > Download As”, including HTML, PDF, Python script, and even LaTeX. This functionality is exposed on the command line through an API called **nbconvert**. It is also possible to execute notebooks within Python scripts, but this is already well documented and the examples below should be equally applicable

**The basic syntax is:**

```
>> jupyter nbconvert --to <format> notebook.ipynb
```

For example, to create a PDF, simply write:

```
>> jupyter nbconvert --to pdf notebook.ipynb
```

By default, nbconvert doesn't execute your notebook code cells.

But if you also wish to, you can specify the **--execute flag**.

```
>> jupyter nbconvert --to pdf --execute notebook.ipynb
```

A common snag arises from the fact that any error encountered running your notebook will halt execution. Fortunately, you can throw in the **--allow-errors flag** to instruct nbconvert to output the error message into the cell output instead.

```
>> jupyter nbconvert --to pdf --execute --allow-errors notebook.ipynb
```