Terminology of Jupyter Notebooks

Excerpt from: https://towardsdatascience.com/jupyter-notebook-for-beginners-a-tutorial-f55b57c23ada

Project Jupyter exists to develop open-source software, open-standards, and services for interactive computing across dozens of programming languages.

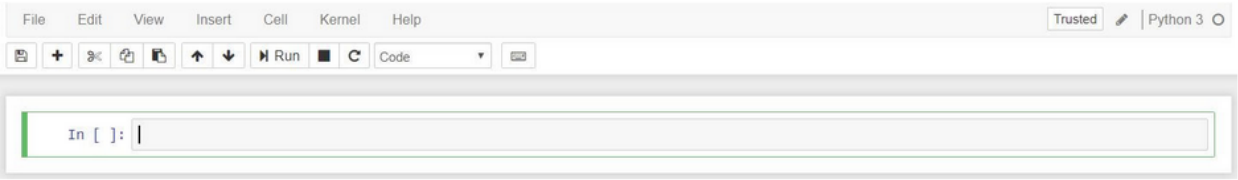
The Jupyter Notebook is an incredibly powerful tool for interactively developing and presenting data science projects. A notebook integrates code and its output into a single document that combines visualizations, narrative text, mathematical equations, and other rich media. The Jupyter project is the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

What is an ipynb File?

It will be useful to understand what a python notebook file really is. Each .ipynb file is a text file that describes the contents of your notebook in a format called JSON. 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 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.

The notebook interface



There are two important terms, 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.

Cells

We’ll return to kernels a little later, but first let’s come to grips with cells. Cells form the body of a notebook. In the screenshot of a new notebook in the section above, that box with the green outline is an empty cell. 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 (see image above).

For example, if you type `print('Hello World!')` into the cell and click the run button in the toolbar above or press `Ctrl + Enter`. The result should look like this:

```
print('Hello World!')
Hello World!
```

When you ran the cell, its output will have been displayed below and the label to its left will have changed from `In [ ]` to `In [1]`. The output of a code cell also forms part of the document, which is why you can see it in this article. You can always tell the difference between code and Markdown cells because code cells have that label on the left and Markdown cells do not. The “In” part of the label is simply short for “Input,” while the label number indicates when the cell was executed on the kernel — in this case the cell was executed first. Run the cell again and the label will change to `In [2]` because now the cell was the second to be run on the kernel. It will become clearer why this is so useful later on when we take a closer look at kernels.

Keyboard shortcuts

Another thing you may observed when running your cells is that their border turned blue, whereas it was green while you are editing. There is always one “active” cell highlighted with a border whose color denotes its current mode, where green means “edit mode” and blue is “command mode.”

So far, we have seen how to run a cell with Ctrl + Enter, but there are plenty more. Keyboard shortcuts are a very popular aspect of the Jupyter environment because they facilitate a speedy cell-based workflow. Many of these are actions you can carry out on the active cell when it’s in command mode.
Below, you’ll find a list of some of Jupyter’s keyboard shortcuts. You’re not expected to pick them up immediately, but the list should give you a good idea of what’s possible.

- Toggle between edit and command mode with Esc and Enter, respectively.
- Once in command mode:
    - Scroll up and down your cells with your Up and Down keys.
    - Press A or B to insert a new cell above or below the active cell.
    - M will transform the active cell to a Markdown cell.
    - Y will set the active cell to a code cell.
    - D + D (D twice) will delete the active cell.
    - Z will undo cell deletion.
    - Hold Shift and press Up or Down to select multiple cells at once.
        - With multple cells selected, Shift + M will merge your selection.
- Ctrl + Shift + -, in edit mode, will split the active cell at the cursor.
- You can also click and Shift + Click in the margin to the left of your cells to select them.

Markdown

Markdown is a lightweight, easy to learn markup language for formatting plain text. Its syntax has a one-to-one correspondance with HTML tags, so some prior knowledge here would be helpful but is definitely not a prerequisite. Let's cover the basics with a quick example.

```
# 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()
And finally, adding images is easy: ![Alt text](https://www.example.com/image.jpg)
```

When attaching images, you have three options:

- Use a URL to an image on the web.
- Use a local URL to an image that you will be keeping alongside your notebook, such as in the same git repo.
- Add an attachment via "Edit > Insert Image"; this will convert the image into a string and store it inside your notebook .ipynb file.
- Note that this will make your .ipynb file much larger!

There is plenty more detail to Markdown, especially around hyperlinking, and it's also possible to simply include plain HTML. Once you find yourself pushing the limits of the basics above, you can refer to the official guide from the creator, John Gruber, on his website.

Kernels

Behind every notebook runs a kernel. When you run a code cell, that code is executed within the kernel and any output is returned back to the cell to be displayed. The kernel's state persists over time and between cells — it pertains to the document as a whole and not individual cells.

For example, if you import libraries or declare variables in one cell, they will be available in another. In this way, you can think of a notebook document as being somewhat comparable to a script file, except that it is multimedia.

Most of the time, the flow in your notebook will be top-to-bottom, but it's common to go back to make changes. In this case, the order of execution stated to the left of each cell, such as In [6], will let you know whether any of your cells have stale output. And if you ever wish to reset things, there are several incredibly useful options from the Kernel menu:

- Restart: restarts the kernel, thus clearing all the variables etc that were defined.
- Restart & Clear Output: same as above but will also wipe the output displayed below your code cells.
- Restart & Run All: same as above but will also run all your cells in order from first to last.

If your kernel is ever stuck on a computation and you wish to stop it, you can choose the Interupt option.

Choosing a kernel

You may have noticed that Jupyter gives you the option to change kernel, and in fact some environments provide many different options to choose from. Back when you created a new notebook from the dashboard by selecting a Python version, you are actually choosing which kernel to use.

Not only are there kernels for different versions of Python, but also for over 100 languages including Java, C, and even Fortran. Data scientists may be particularly interested in the kernels for R and Julia, as well as both imatlab and the Calysto MATLAB Kernel for Matlab. The SoS kernel provides multi-language support within a single notebook. Each kernel has its own installation instructions, but will likely require you to run some commands on your computer.