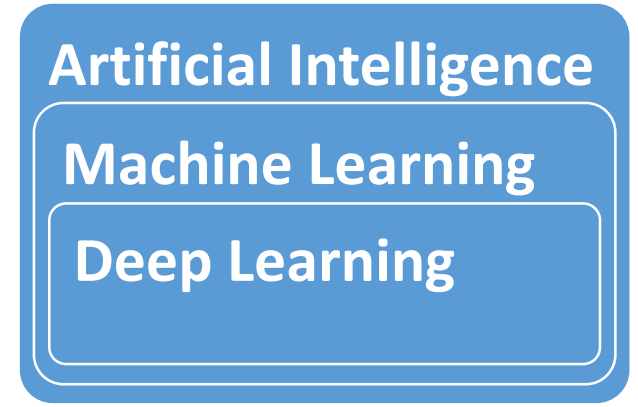
Python-Jupyter Notebook

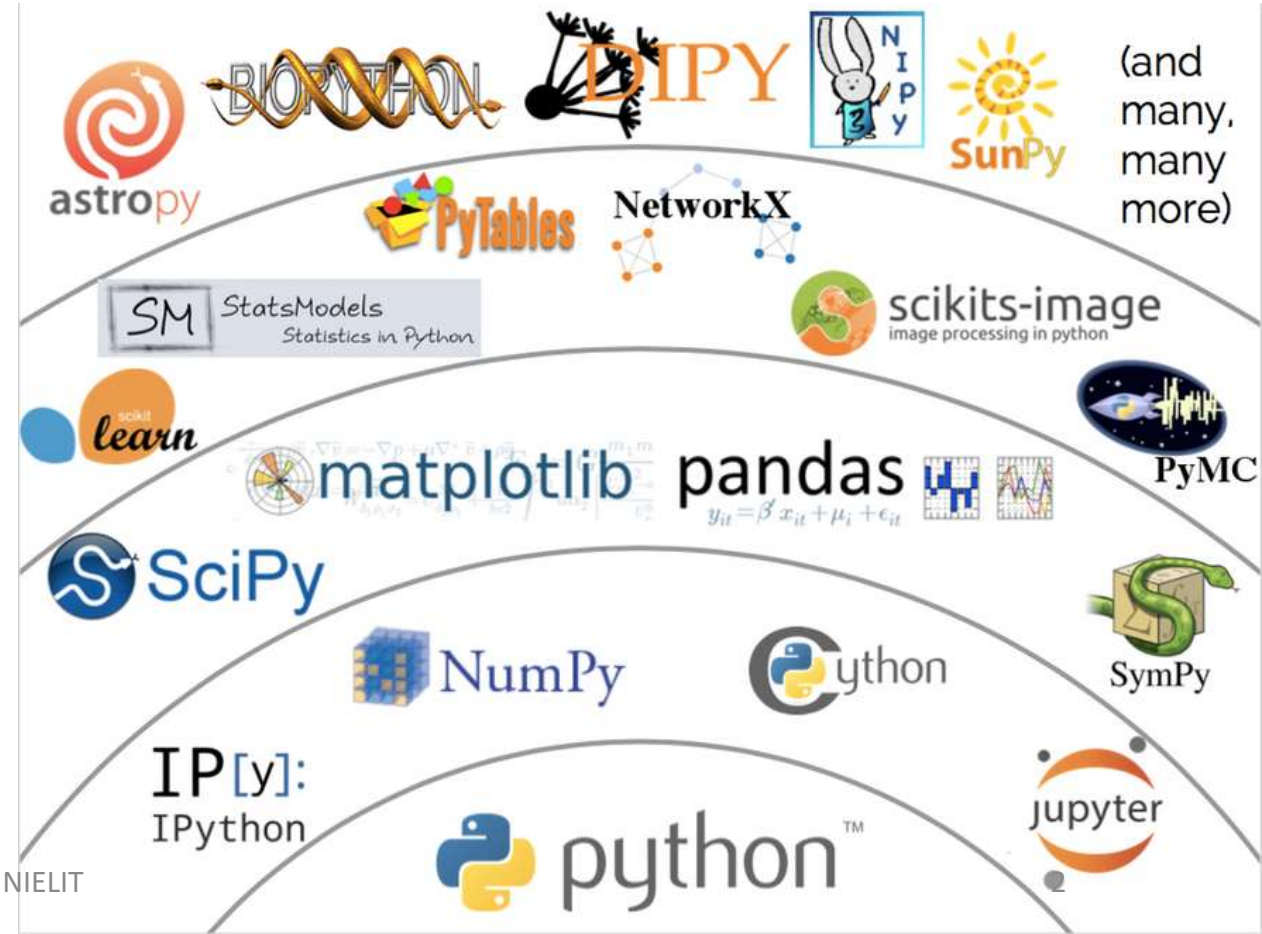
Dr. Sarwan Singh

NIELIT Chandigarh

Agenda

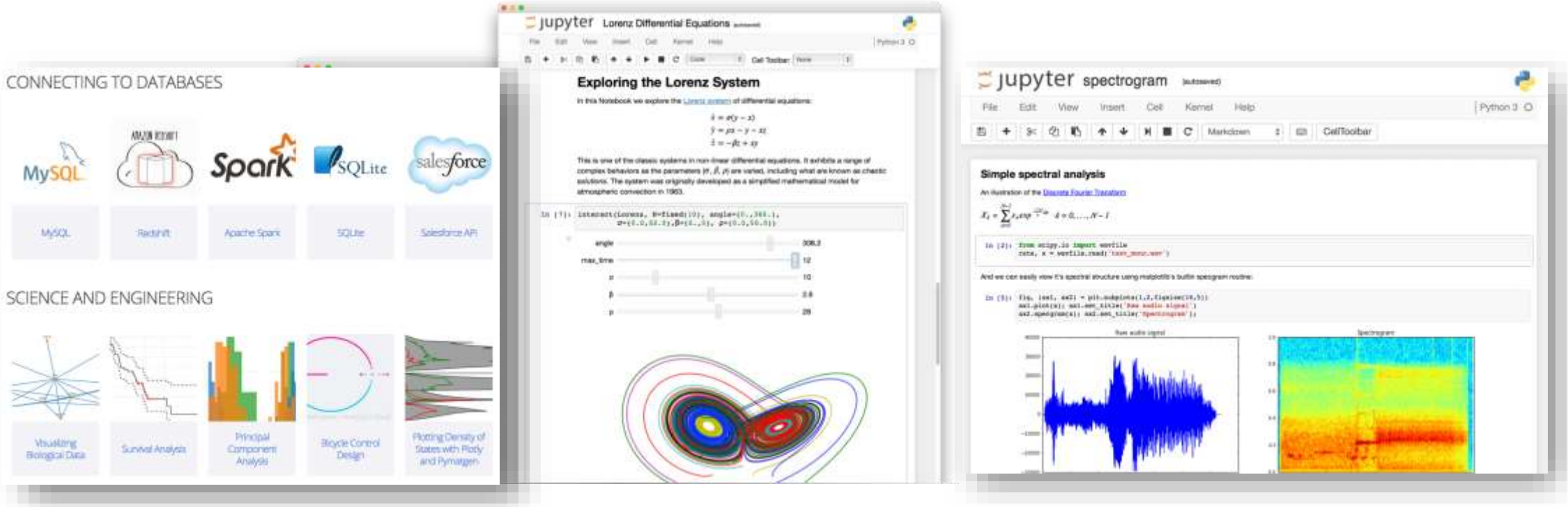
- Introduction
- History and usage
- IPYNB file
- Tips To Effectively & Efficiently Use Jupyter Notebooks
- How Jupyter Works
- Jupyter and The IPython Kernel





"Jupyter" is a loose acronym meaning

Julia, Python, and R



Language of choice

The Notebook has support for over 40 programming languages, including Python, R, Julia, and Scala.

Share notebooks

Notebooks can be shared with others using email, Dropbox, GitHub and the Jupyter Notebook Viewer.

Interactive output

Your code can produce rich, interactive output: HTML, images, videos, LaTeX, and custom MIME types.

Big data integration

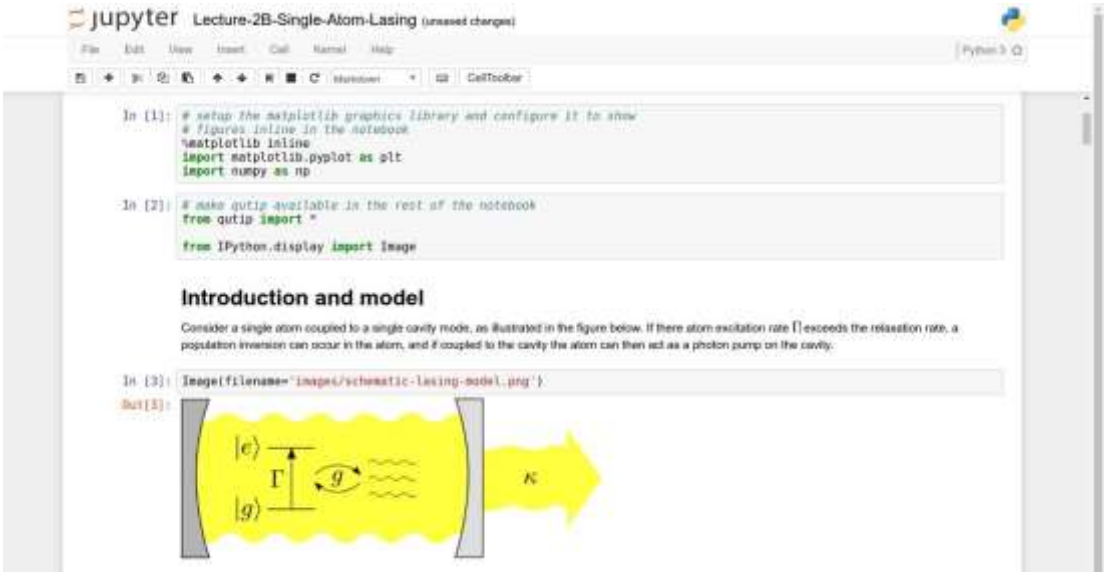
Leverage big data tools, such as Apache Spark, from Python, R and Scala. Explore that same data with pandas, scikit-learn, ggplot2, TensorFlow.

Introduction

- The **Jupyter Notebook** is an open-source web application that allows to create and share documents that contain

  live code, equations, visualizations and narrative text.

- Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.



Source : jupyter.org

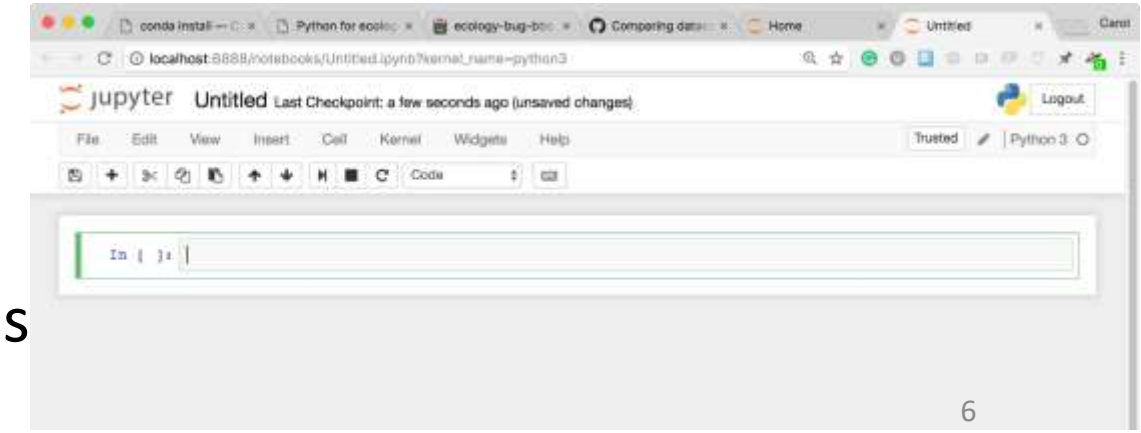
Jupyter Notebook App

- As a server-client application, the Jupyter Notebook App allows you to edit and run your notebooks via a web browser.
- The application can be executed on a PC without Internet access or it can be installed on a remote server, where you can access it through the Internet.
- Two main components are the kernels and a dashboard.
  - A kernel is a program that runs and introspects the user's code. The Jupyter Notebook App has a kernel for Python code, but there are also kernels available for other programming languages.
  - The dashboard of the application not only shows you the notebook documents that you have made and can reopen but can also be used to manage the kernels: you can which ones are running and shut them down if necessary.

History

- Late 1980s, Guido Van Rossum begins to work on Python at the National Research Institute for Mathematics and Computer Science in the Netherlands
- late 2001, twenty years later. Fernando Pérez starts developing IPython.
- In 2005, both Robert Kern and Fernando Pérez attempted building a notebook system. Unfortunately, the prototype had never become fully usable.
- In 2007, they formulated another attempt at implementing a notebook-type system
- In the summer of 2011, the prototype of

  web notebook was incorporated
- In subsequent years, the team got awards

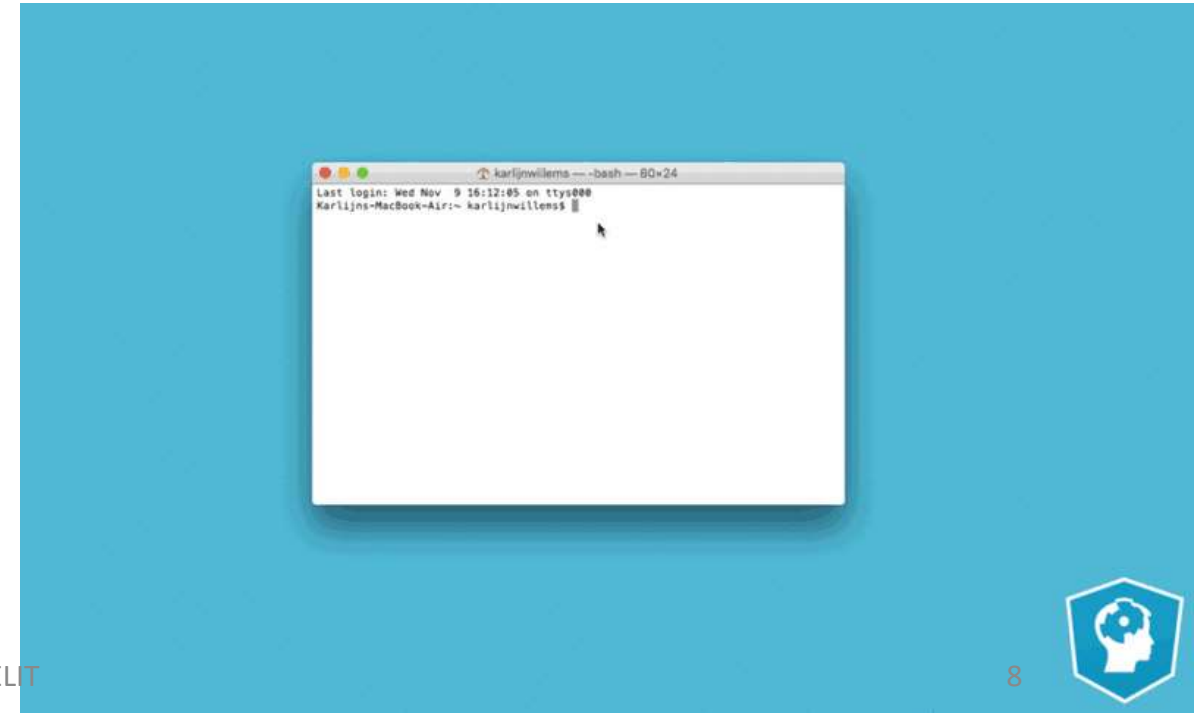


History

- in 2014, Project Jupyter started as a spin-off project from IPython. IPython is now the name of the Python backend, which is also known as the kernel.
- Other products similar to notebook
  - Sage notebook was based on the layout of Google notebooks
  - Mathematica notebooks and Maple worksheets. The Mathematica notebooks were created as a front end or GUI in 1988 by Theodore Gray.

Jupyter Notebooks With The Anaconda Python Distribution

- Use the Anaconda distribution to install both Python and the notebook application.
- The advantage of Anaconda is that you have access to over 720 packages that can easily be installed with Anaconda's conda, a package, dependency, and environment manager.



Source : datacamp.com

IPYNB file

- An **IPYNB file** is a notebook document used by Jupyter Notebook, an interactive computational environment designed to help scientists work with the Python language and their data. ...
- NOTE: Jupyter notebooks were formerly known as IPython notebooks, which is where the "**ipynb**" extension got its name.

Tips To Effectively & Efficiently Use Jupyter Notebooks

- provide comments and documentation to code.
- consistent naming scheme, code grouping, limit your line length, ...
- Don't forget to name your notebook documents!
- Try to keep the cells of your notebook simple: don't exceed the width of your cell and make sure that you don't put too many related functions in one cell.
- If possible, import your packages in the first code cell of your notebook
- Display the graphics inline. The magic command %matplotlib inline will definitely come in handy to suppress the output of the function on a final line.
- Don't forget to add a semicolon to suppress the output and to just give back the plot itself.
- magic commands such as %run to execute a whole Python script, in a notebook cell.

Lesser known ways

- By running % lsmagic in a cell you get a list of all the available magics.
- use % to start a single-line expression to run with the magics command.
- Or you can use a double %% to run a multi-line expression.
- % env to list your environment variables.
- !: to run a shell command. E.g., ! pip freeze | grep pandas to see what version of pandas is installed.
- % matplotlib inline to show matplotlib plots inline the notebook.
- % pastebin 'file.py' to upload code to pastebin and get the url returned.
- % bash to run cell with bash in a subprocess.

Lesser known ways

%time will time whatever you evaluate
%%latex to render cell contents as LaTeX

```
In [35]: %%latex
\begin{align}
a = \frac{1}{2} && b = \frac{1}{3} && c = \frac{1}{4} \\
a && b && c \\
1 && 2 && 3
\end{align}
```

$$a = \frac{1}{2} \qquad b = \frac{1}{3} \qquad c = \frac{1}{4}$$
$$a \qquad\quad b \qquad\quad c$$
$$1 \qquad\quad 2 \qquad\quad 3$$

%timeit will time whatever you evaluate multiple times and give you the best, and the average times

```
%time x = range(10000)
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 316 µs
```

```
%%timeit x = range(10000)
   max(x)
```

```
1000 loops, best of 3: 344 µs per loop
```

Lesser known ways

- %prun, %lprun, %mprun can give you line-by-line breakdown of time and memory usage in a function or script. See a good tutorial here.
- %% HTML: to render the cell as HTML. So you can even embed an image or other media in your notebook:

```
In [22]: %%HTML
         <img src="http://techiegiveways.com/wp-content/uploads/2015
         <h1>Jupyter Notebooks can render HTML</h1>
```

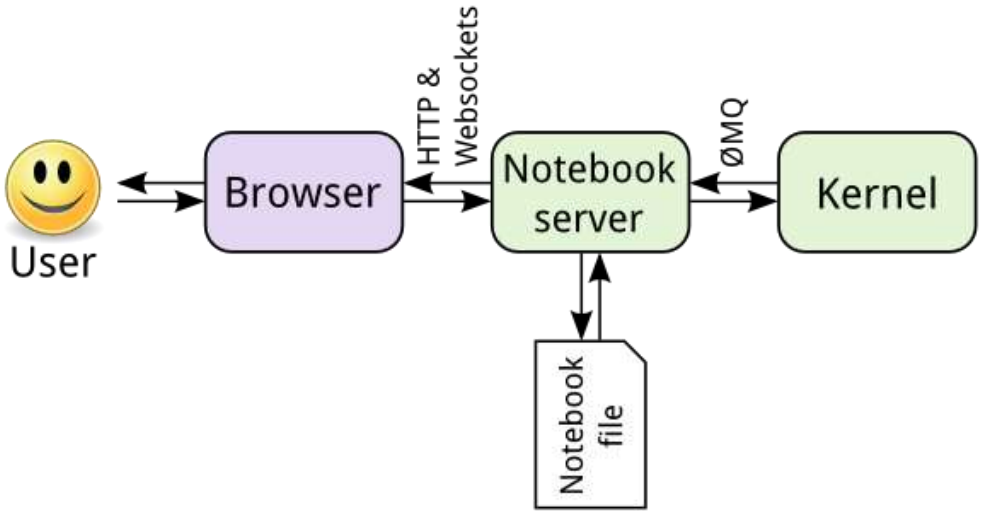
Jupyter Notebooks can render HTML

https://hub.mybinder.org/user/ipython-ipython-in-depth-gymrx60d/notebooks/binder/Index.ipynb
https://help.github.com/articles/basic-writing-and-formatting-syntax/
https://www.dataquest.io/blog/jupyter-notebook-tips-tricks-shortcuts/

Notebooks

- The Notebook frontend does something extra. In addition to running your code, it stores code and output, together with markdown notes, in an editable document called a notebook. When you save it, this is sent from your browser to the notebook server, which saves it on disk as a JSON file with a .ipynb extension.

The notebook server, not the kernel, is responsible for saving and loading notebooks, so you can edit notebooks even if you don't have the kernel for that language—you just won't be able to run code. The kernel doesn't know anything about the notebook document: it just gets sent cells of code to execute when the user runs them.



How Jupyter works

..But first, Terminal IPython

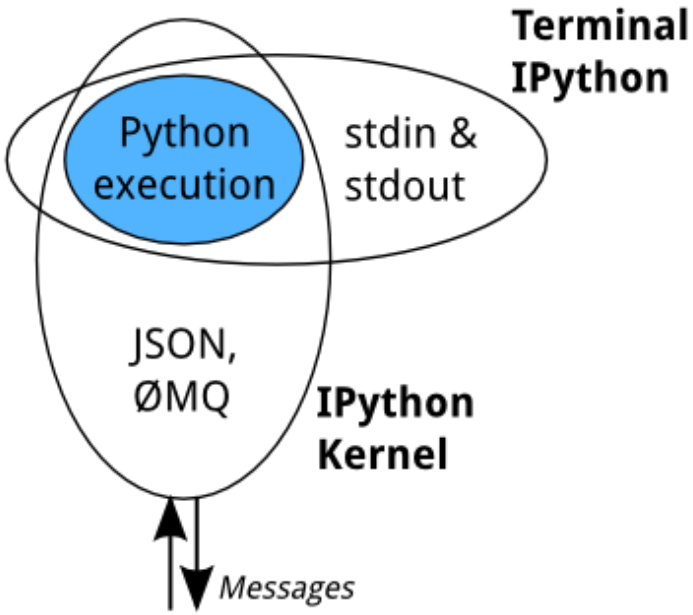
- When you type ipython, you get the original IPython interface, running in the terminal. It does something like this:

```
while True:
    code = input(">>> ")
    exec(code)
```

- Of course, it's much more complex, because it has to deal with multi-line code, tab completion using readline or prompt-toolkit, magic commands, and so on.
- But the model is like that: prompt the user for some code, and when they've entered it, exec it in the same process. This model is often called a REPL, or Read-Eval-Print-Loop.

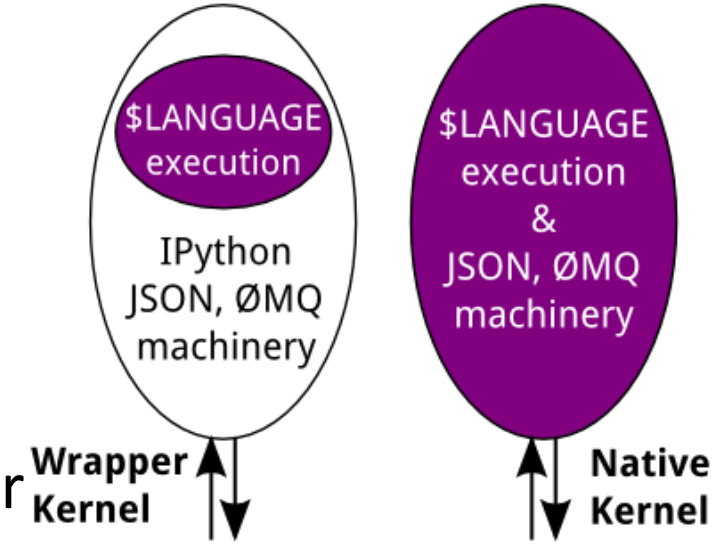
Jupyter and The IPython Kernel

- Jupyter expands on this REPL model and provides other interfaces to running code–the notebook, the Qt console, jupyter console in the terminal, and third party interfaces—use the IPython Kernel.
- This is a separate process which is responsible for running user code, and things like computing possible completions. Frontends communicate with it using JSON messages sent over ZeroMQ sockets; the protocol they use is described in the jupyter protocol.



- The core execution machinery for the kernel is shared with terminal IPython

- A kernel process can be connected to more than one frontend simultaneously. In this case, the different frontends will have access to the same variables.
- This design was intended to allow easy development of different frontends based on the same kernel, but it also made it possible to support new languages in the same frontends, by developing kernels in those languages
- Today, there are two ways to develop a kernel for another language.
  - Wrapper kernels reuse the communications machinery from IPython, and implement only the core execution part.
  - Native kernels implement execution and communications in the target language
- Wrapper kernels are easier to write quickly for languages that have good Python wrappers, like octave_kernel, or languages where it's impractical to implement the communications machinery, like bash_kernel. Native kernels are likely to be better maintained by the community using them, like IJulia or IHaskell.

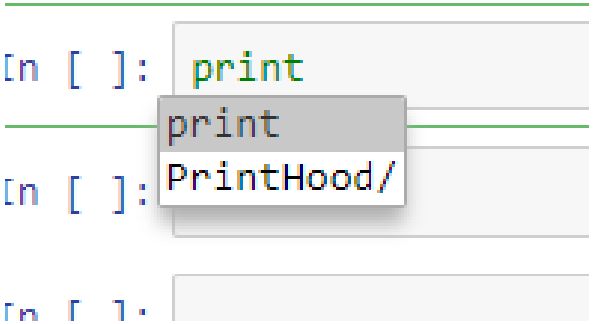


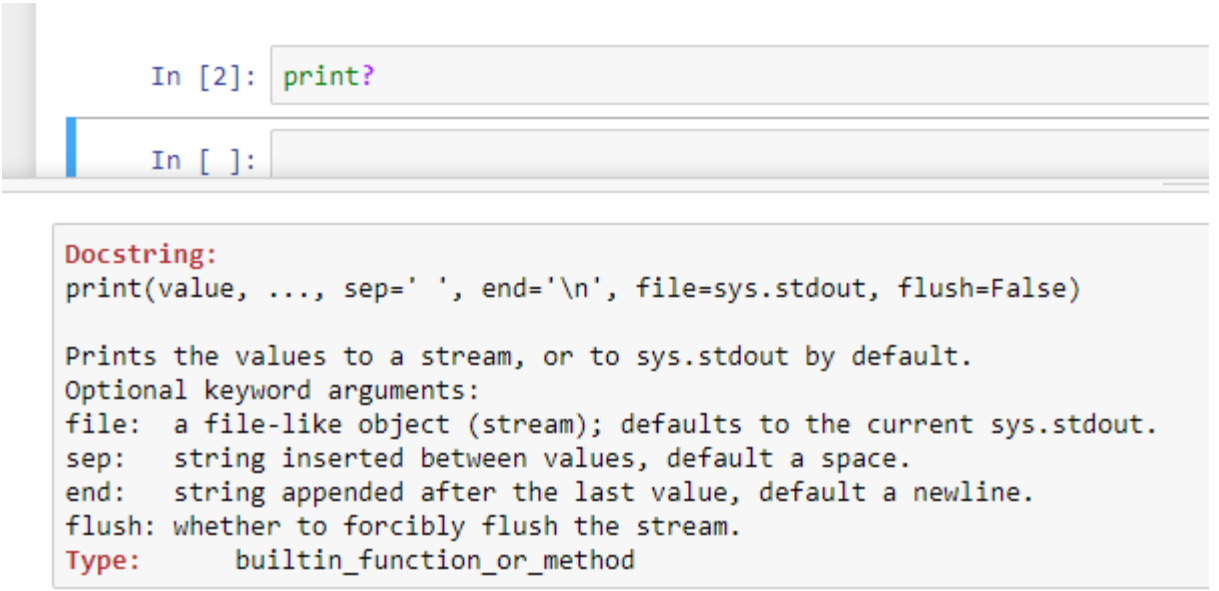
Modal Editing

- Modal Editing means that the effect of typing at the keyboard depends on mode. The two modes are:
  - **Edit Mode:**
    - Indicated by a green border around the cell
    - Whatever you type appears as in the cell
  - **Command Mode:**
    - The green border is replaced by grey border
    - Keystrokes are interpreted as commands: example typing b adds a new cell below the current one
  - **Switching between modes** :
    - to command mode from edit mode, press Esc key or CTRL M
    - To edit mode from command mode, hit enter or click on a cell

Working with Notebooks

- Press tab to get help
- ? for help
- Shift+Enter to execute command in current cell
- %run test.py to run python file





Test.py

```
for i in range(5):
    print('Jai Ho')
```

Cell Magic

```
In [15]: %%file prog1.py
         for i in range(5):
             print('Jai Ho')

         Writing prog1.py

In [16]: run prog1.py

         Jai Ho
         Jai Ho
         Jai Ho
         Jai Ho
         Jai Ho
```

```
In [3]: pwd

Out[3]: 'C:\\Users\\Electronics'

In [4]: ls

 Volume in drive C is Windows
 Volume Serial Number is EE7B-697C

 Directory of C:\Users\Electronics

18-06-2018  22:14    <DIR>          .
18-06-2018  22:14    <DIR>          ..
18-06-2018  10:02    <DIR>          .anaconda
02-04-2018  17:29    <DIR>          .AndroidStudio3.1
18-06-2018  10:05    <DIR>          .conda
18-06-2018  10:04                43 .condarc
17-06-2018  23:54               185 .gitconfig
03-04-2018  11:32    <DIR>          .gradle
18-06-2018  10:19             3,047 ST401.ipynb
18-06-2018  22:19                36 test.py
03-05-2018  12:43    <DIR>          tutorials_jupyter
03-05-2018  00:42           429,144 tutorials_jupyter.zi
18-06-2018  22:22             8,216 Untitled.ipynb
09-05-2018  18:12           137,858 Untitled1.ipynb
17-06-2018  01:51    <DIR>          Videos
              16 File(s)      5,477,489 bytes
              39 Dir(s)  258,649,706,496 bytes free

In [14]: run test.py

         Jai Ho
         Jai Ho
         Jai Ho
         Jai Ho
         Jai Ho
```