



PyNb: Jupyter Notebooks as plain Python code

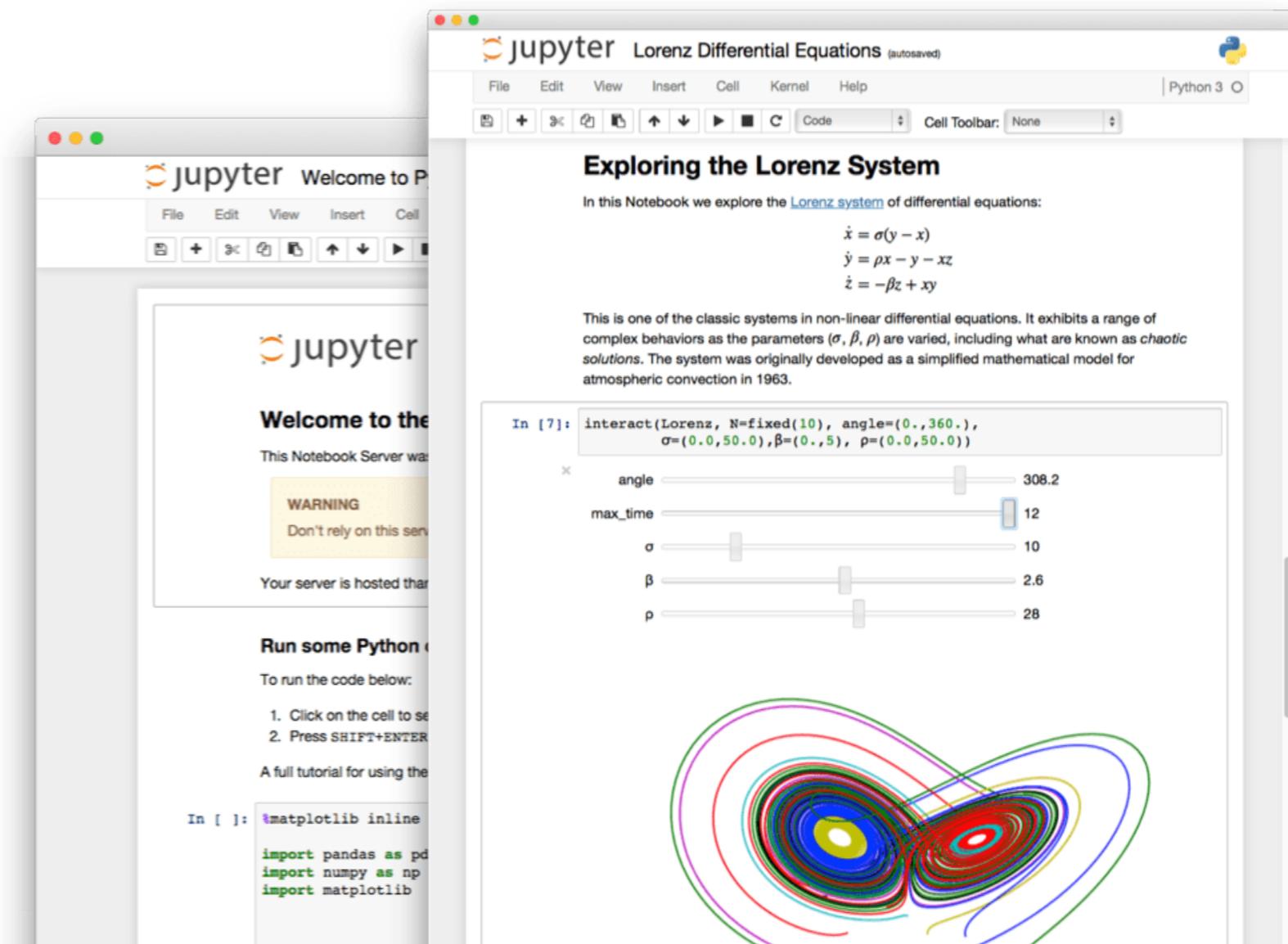
Michele Dallachiesa

michele.dallachiesa@minodes.com

```
$ git clone https://github.com/minodes/pynb
$ pip install pynb
```

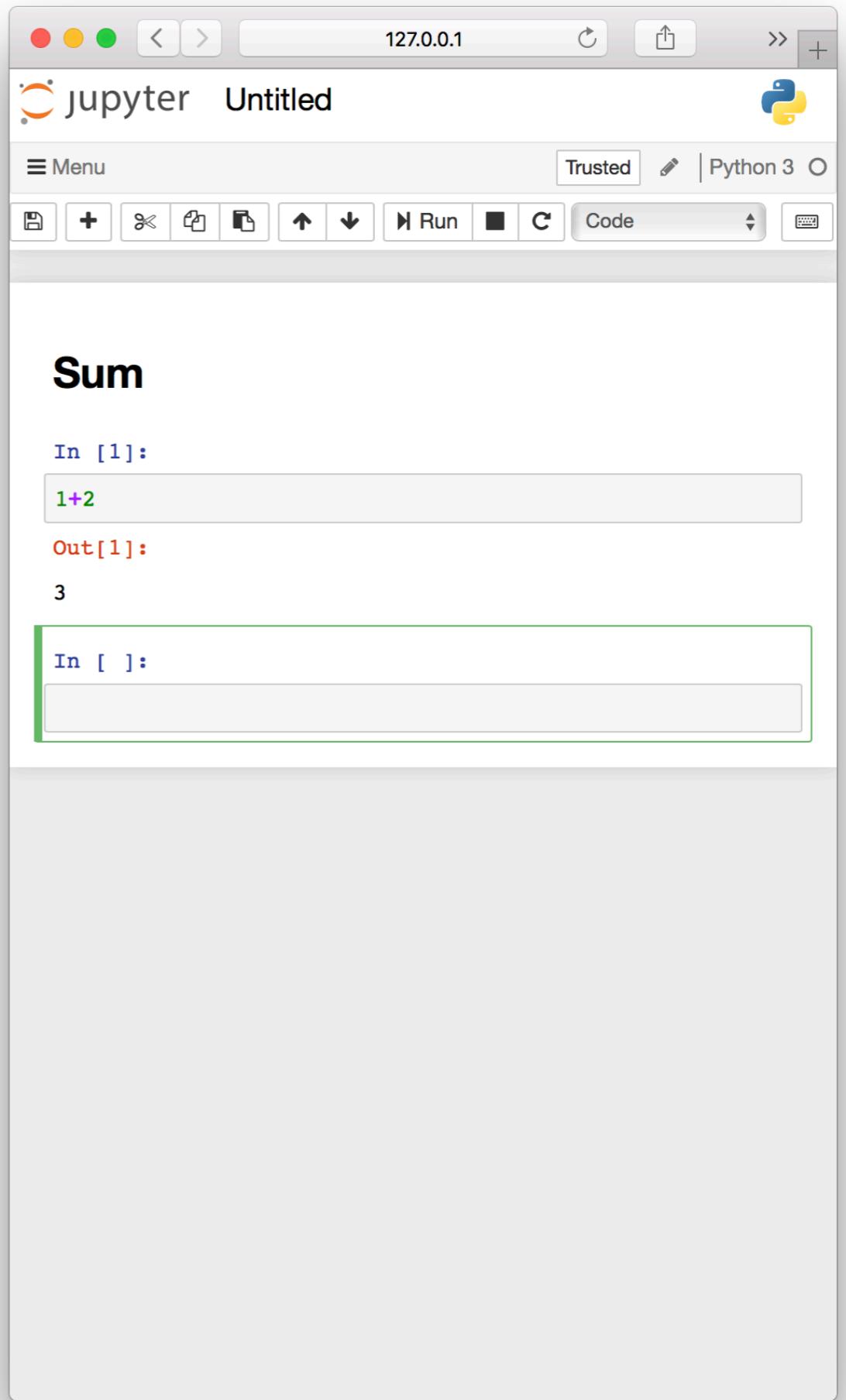
Jupyter notebooks

- Documents containing Live code, visualisations and narrative text



Interactive computing

Experiment with algorithms
and data in a reproducible
and shareable way



The screenshot shows a Jupyter Notebook window titled "jupyter Untitled". The title bar includes the IP address "127.0.0.1", a refresh icon, and a Python 3 logo. The menu bar has "Menu", "Trusted", "Python 3", and "Code" options. Below the menu is a toolbar with icons for file operations like save, new, and delete, and navigation like up, down, and run. The main area is titled "Sum" and contains a code cell labeled "In [1]:" with the expression "1+2" and an output cell labeled "Out[1]:" with the result "3". A green border highlights the "In []:" input field at the bottom.

```
In [1]:  
1+2  
Out[1]:  
3  
In [ ]:
```

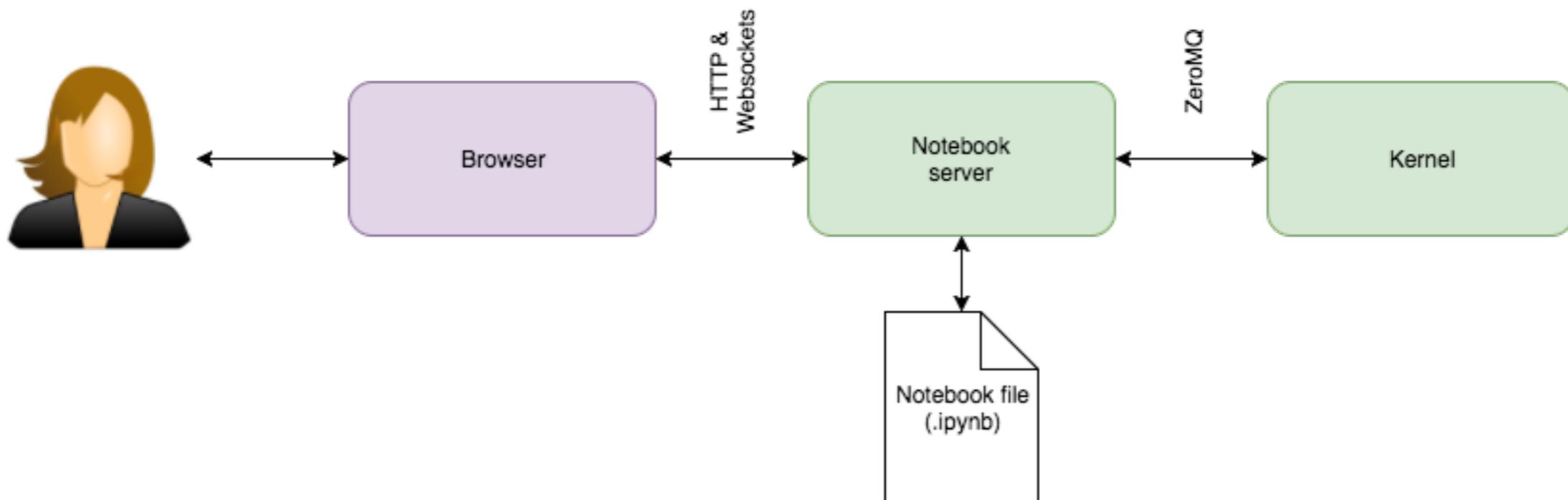
Shareable

Open document format based on JSON

```
1  {
2    "cells": [
3      { "cell_type": "markdown",
4        "metadata": {},
5        "source": [ "# Sum" ] },
6      { "cell_type": "code",
7        "execution_count": 1,
8        "metadata": {},
9        "outputs": [ {
10          "data": { "text/plain": [ "3" ] },
11          "execution_count": 1,
12          "metadata": {},
13          "output_type": "execute_result" } ],
14        "source": [ "1+2" ] },
15      { "cell_type": "code",
16        "execution_count": null,
17        "metadata": {},
18        "outputs": [],
19        "source": [ ] } ],
20      "metadata": {
21        "kernelspec": {
22          "display_name": "Python 3",
23          "language": "python",
24          "name": "python3"
25        },
26        "language_info": {
27          "codemirror_mode": {
28            "name": "ipython",
29            "version": 3
30          },
31          "file_extension": ".py",
32          "mimetype": "text/x-python",
33          "name": "python",
34          "nbconvert_exporter": "python",
35          "pygments_lexer": "ipython3",
36          "version": "3.6.1"
37        }
38      }, "nbformat": 4, "nbformat_minor": 2
39    }
```

How does it work?

- **User** interacts with **browser**, **notebook-server** bridges instructions to **kernel**
- **Kernel** provides “computing service”



Strengths

- **Interactive and visual computing** based on open standards for all major programming languages
- **Widely popular in data science community**
- **Ideal for light exploration of APIs and data**, data cleaning and transformation, statistical modeling, data visualisation, machine learning

Limitations

- **No version control:** JSON diffs require application-specific interpretation
- **Encourages unstructured code:** code duplication, no modules, flat code organisation
- **Uncertain execution state:** re-execution of cells
- **No IDE features:** limited autocompletion, no code navigation, refactoring, style compliance

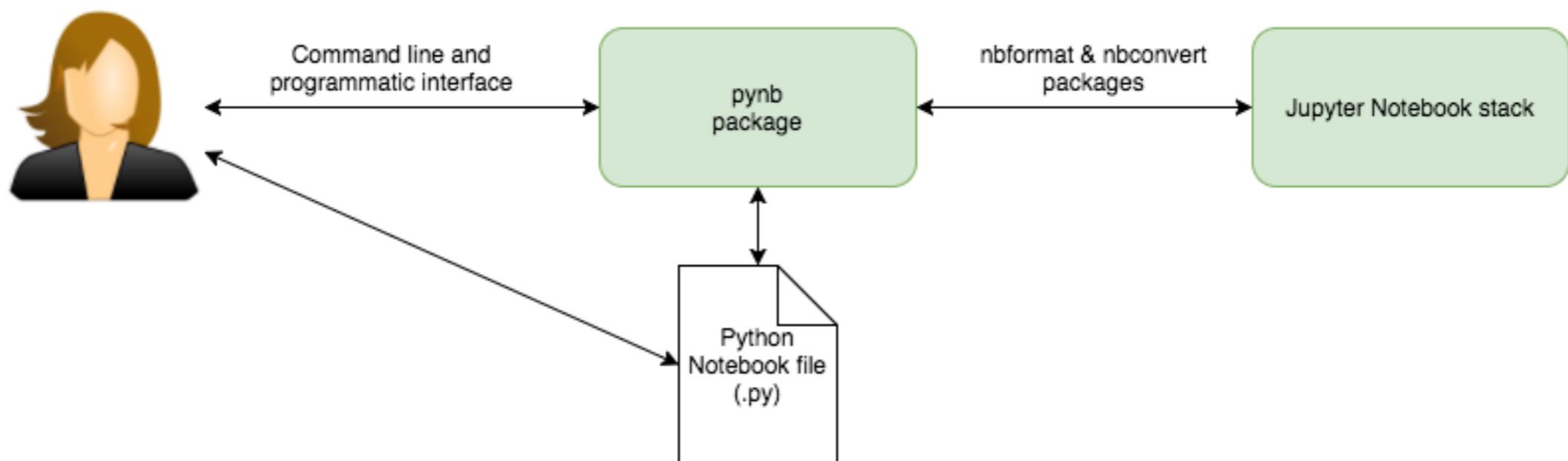
Solution: Python Notebooks

- Jupyter Notebooks as plain Python code
- Enables Python IDE/editors, version control, avoids inconsistent execution state

```
def cells(a, b):
    ...
    # Sum
    ...
    a, b = int(a), int(b)
    ...
    ...
    a + b
```

PyNb package

- **User** interacts with **Python IDE/editor** and **PyNb**
- **PyNb** bridge to **Jupyter Notebook stack**



PyNb features

- **Supports Python and Jupyter notebooks:**
Execution and conversion
- **Command-line and programmatic interfaces:**
Fine-grained control on parameters and execution
- **Transparent caching system for cell execution:**
Cache database queries, processing results, ...

Command-line interface

```
$ pynb sum.py --param a=3 --param b=5 --export-ipynb sum.ipynb
```

```
def cells(a, b):
    ...
    # Sum
    ...
    a, b = int(a), int(b)
    ...
    ...
    a + b
```

The screenshot shows a Jupyter Notebook interface with the following details:

- Title:** jupyter sum
- Kernel:** Python 3
- In [1]:** `# Parameters:
b = '5'
a = '3'`
- In [2]:** `a, b = int(a), int(b)`
- In [3]:** `a + b`
- Out[3]:** 8
- Summary:**
 - Notebook class name: Notebook
 - Notebook cells name: notebooks/sum.py
 - Execution time: 2017-12-08 11:52:15.221826
 - Execution duration: 1.57s
 - Command line: ['pynb', 'notebooks/sum.py', '--param', 'a=3', '--param', 'b=5', '--export-ipynb', 'notebooks/sum.ipynb']

Programmatic interface

```
$ python3 sumapp.py --b 3 --export-ipynb sumapp.ipynb
```

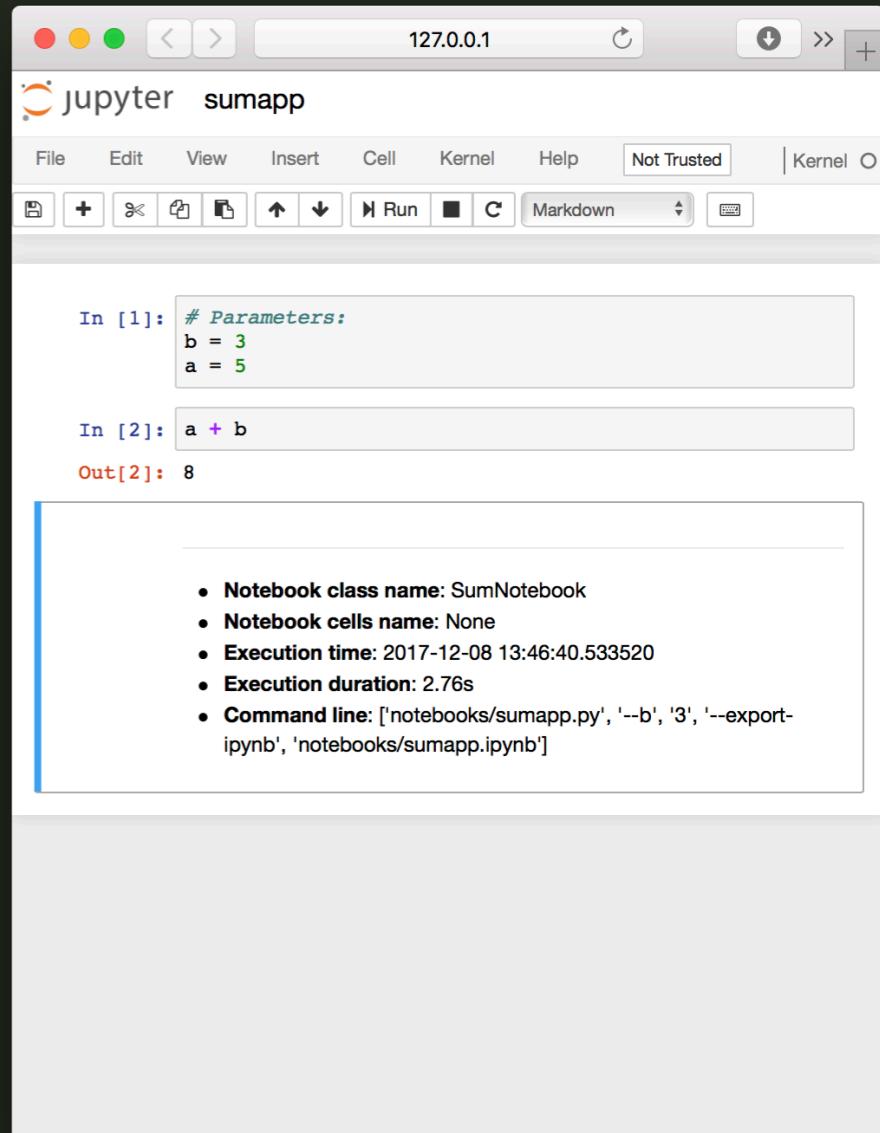
```
from pynb.notebook import Notebook

class SumNotebook(Notebook):
    def cells(self, a, b):
        a + b

if __name__ == "__main__":
    nb = SumNotebook()
    nb.add_argument('--a', default=5, type=int)
    nb.add_argument('--b', type=int)
    nb.add_argument('--print-ipynb',
                    action="store_true", default=False)

    nb.run()

    if nb.args.print_ipynb:
        nb.export_ipynb('-')
```



The screenshot shows a Jupyter Notebook interface with the title "jupyter sumapp". The notebook has two cells:

- In [1]: `# Parameters:
b = 3
a = 5`
- In [2]: `a + b`

The output for In [2] is Out[2]: 8.

At the bottom of the interface, there is a summary of the session:

- Notebook class name: SumNotebook
- Notebook cells name: None
- Execution time: 2017-12-08 13:46:40.533520
- Execution duration: 2.76s
- Command line: ['notebooks/sumapp.py', '--b', '3', '--export-ipynb', 'notebooks/sumapp.ipynb']

Cached cell execution

- Caching system avoids re-evaluation of cells, saving computation time

First execution

Cell 1: rows = db.query(...)

Cell 2: data = clean(rows)

Cell 3: len(data)

Cached cell execution

- Caching system avoids re-evaluation of cells, saving computation time

First execution

Cell 1: rows = db.query(...)
Execution time: 500s

Cell 2: data = clean(rows)

Cell 3: len(data)

Cached cell execution

- Caching system avoids re-evaluation of cells, saving computation time

First execution

Cell 1: rows = db.query(...)
Execution time: 500s

Cell 2: data = clean(rows)
Execution time: 80s

Cell 3: len(data)

Cached cell execution

- Caching system avoids re-evaluation of cells, saving computation time

First execution

Cell 1: rows = db.query(...)
Execution time: 500s

Cell 2: data = clean(rows)
Execution time: 80s

Cell 3: len(data)
Execution time: 20s

Total execution time: 600s

Cached cell execution

- Caching system avoids re-evaluation of cells, saving computation time

First execution

Cell 1: rows = db.query(...)
Execution time: 500s

Cell 2: data = clean(rows)
Execution time: 80s

Cell 3: len(data)
Execution time: 20s

Total execution time: 600s

Second execution

Cell 1: rows = db.query(...)

Cell 2: data = **filter**(rows)

Cell 3: len(data)

Cached cell execution

- Caching system avoids re-evaluation of cells, saving computation time

First execution

Cell 1: rows = db.query(...)
Execution time: 500s

Cell 2: data = clean(rows)
Execution time: 80s

Cell 3: len(data)
Execution time: 20s

Total execution time: 600s

Second execution

Cell 1: rows = db.query(...)
Execution time: 1s

Cell 2: data = **filter**(rows)

Cell 3: len(data)

Cached cell execution

- Caching system avoids re-evaluation of cells, saving computation time

First execution

Cell 1: rows = db.query(...)
Execution time: 500s

Cell 2: data = clean(rows)
Execution time: 80s

Cell 3: len(data)
Execution time: 20s

Total execution time: 600s

Second execution

Cell 1: rows = db.query(...)
Execution time: 1s

Cell 2: data = filter(rows)
Execution time: 80s

Cell 3: len(data)

Cached cell execution

- Caching system avoids re-evaluation of cells, saving computation time

First execution

Cell 1: rows = db.query(...)
Execution time: 500s

Cell 2: data = clean(rows)
Execution time: 80s

Cell 3: len(data)
Execution time: 20s

Total execution time: 600s

Second execution

Cell 1: rows = db.query(...)
Execution time: 1s

Cell 2: data = filter(rows)
Execution time: 80s

Cell 3: len(data)
Execution time: 20s

Total execution time: 101s

Conclusion

- **Jupyter notebook** interactive computation, ideal to experiment, hard to maintain
- **Python notebook** regular Python code, ideal for templating and reporting tasks, consolidation of Jupyter notebooks
- **PyNb** bridge between Jupyter and Python notebook formats



Thank You!

(We're Hiring!)

Michele Dallachiesa

michele.dallachiesa@minodes.com

```
$ git clone https://github.com/minodes/pynb
$ pip install pynb
```