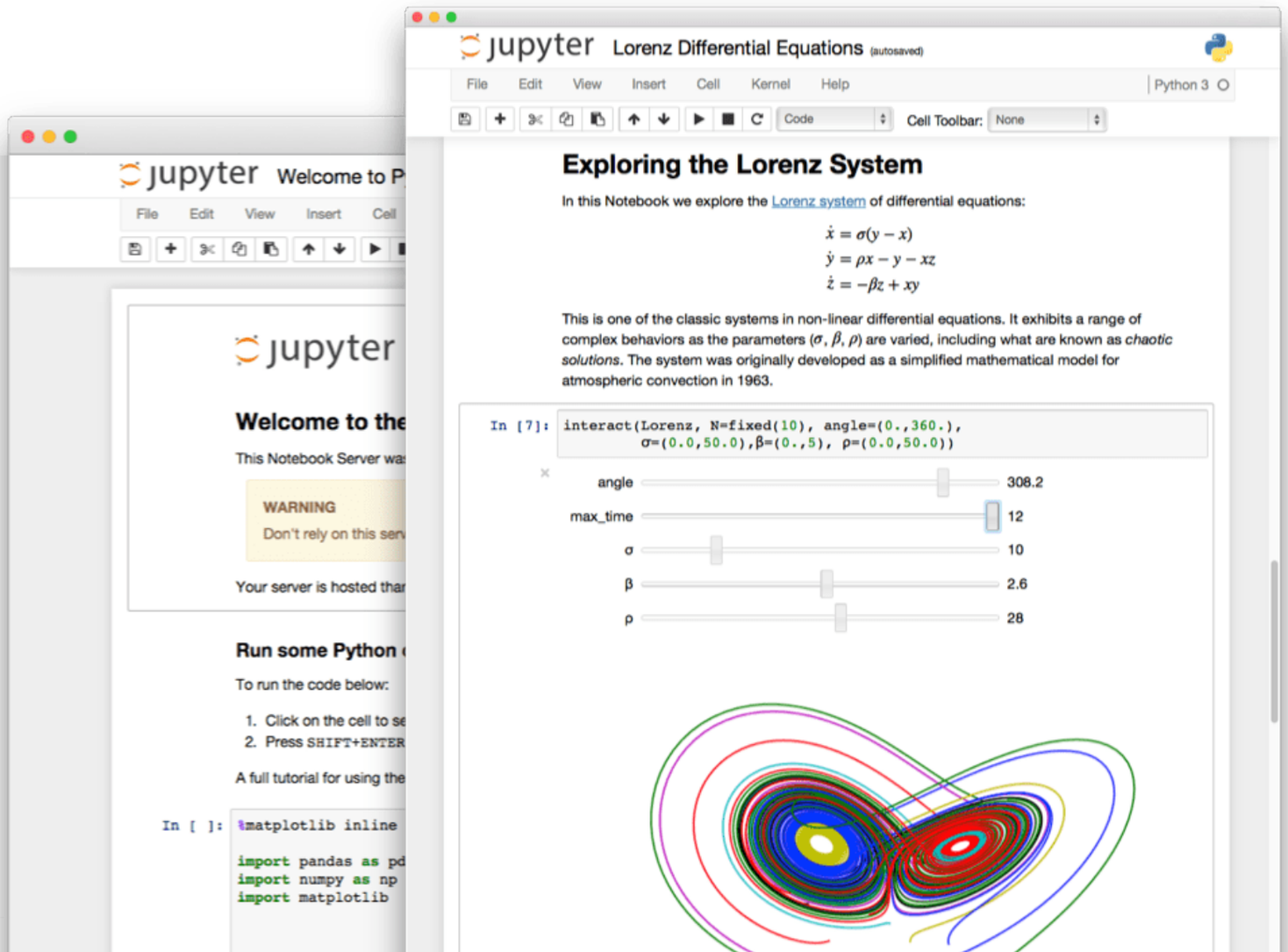
PyNb: Jupyter Notebooks as plain Python code

Michele Dallachiesa
michele.dallachiesa@minodes.com

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

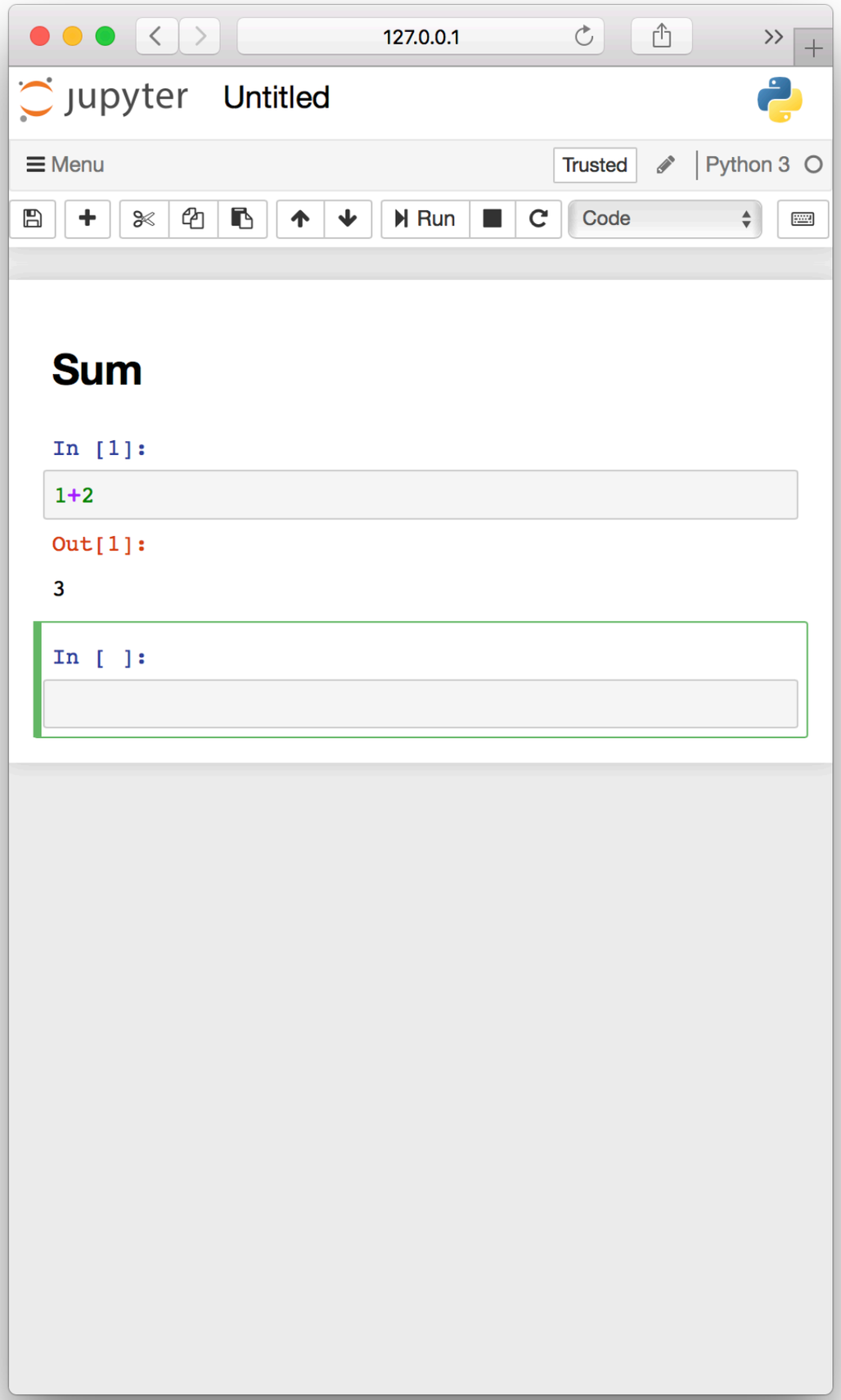
Jupyter notebooks

- Documents containing Live code, visualisations and narrative text



Interactive computing

Experiment with algorithms and data in a reproducible and shareable way



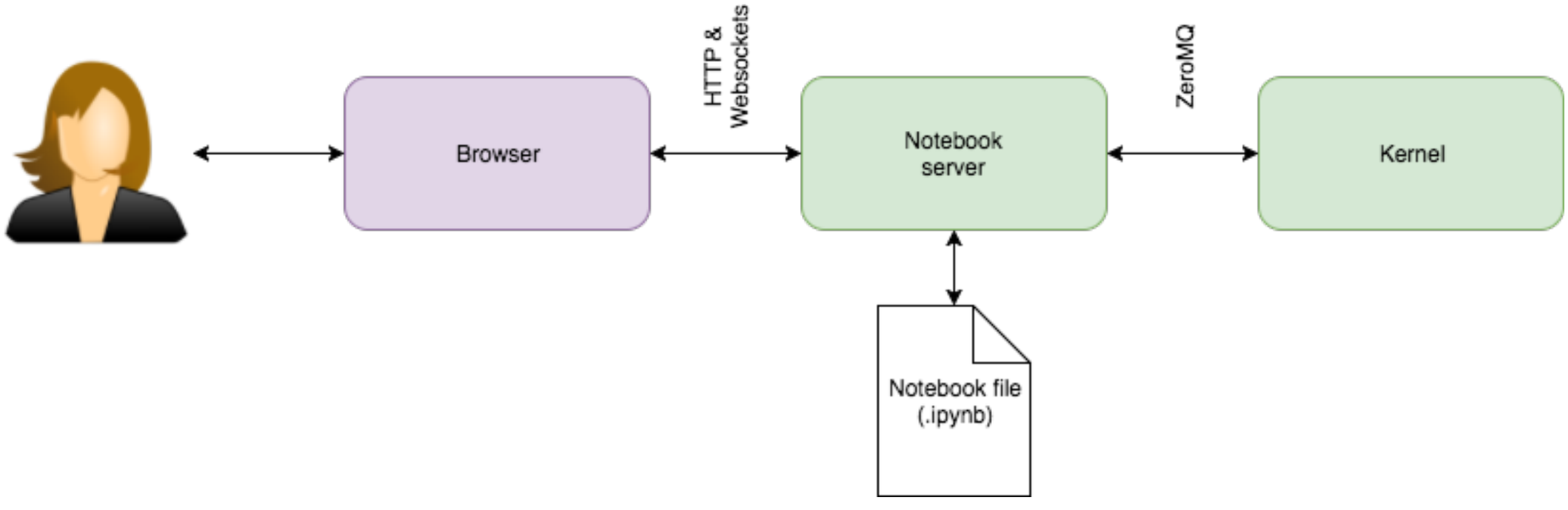
Shareable

Open document format based on JSON


```
{
 "cells": [
  { "cell_type": "markdown",
    "metadata": {},
    "source": [ "# Sum" ] },
  { "cell_type": "code",
    "execution_count": 1,
    "metadata": {},
    "outputs": [ {
      "data": { "text/plain": [ "3" ] },
      "execution_count": 1,
      "metadata": {},
      "output_type": "execute_result" } ],
    "source": [ "1+2" ] },
  { "cell_type": "code",
    "execution_count": null,
    "metadata": {},
    "outputs": [],
    "source": [] } ],
 "metadata": {
  "kernelspec": {
   "display_name": "Python 3",
   "language": "python",
   "name": "python3"
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython",
    "version": 3
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.6.1"
  }
 }, "nbformat": 4, "nbformat_minor": 2
}
```

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

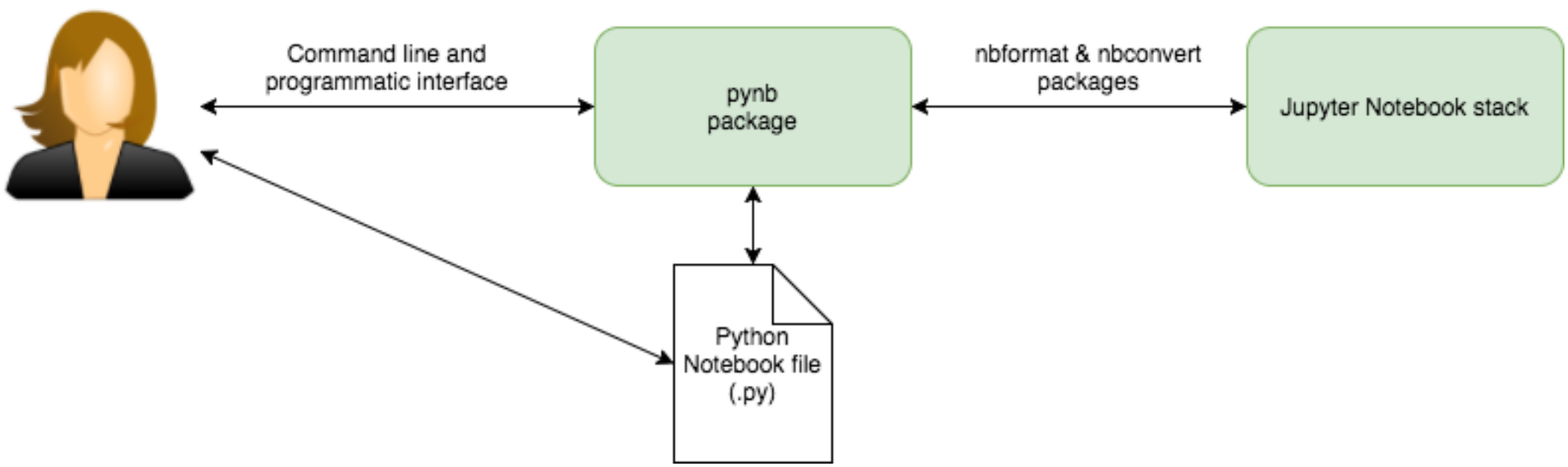
Solution: **Python Notebooks**

- 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

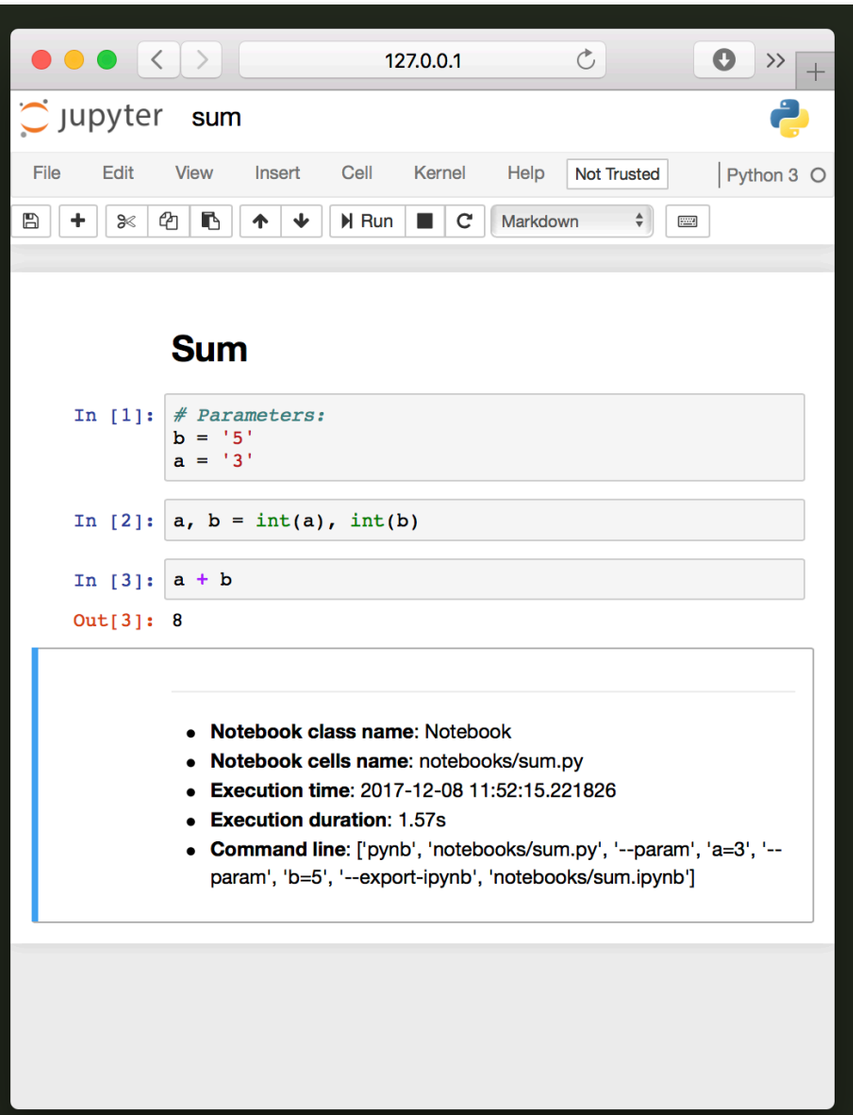
- **Command-line and programmatic interfaces:**
  Fine-grained control on parameters and execution

- **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
```

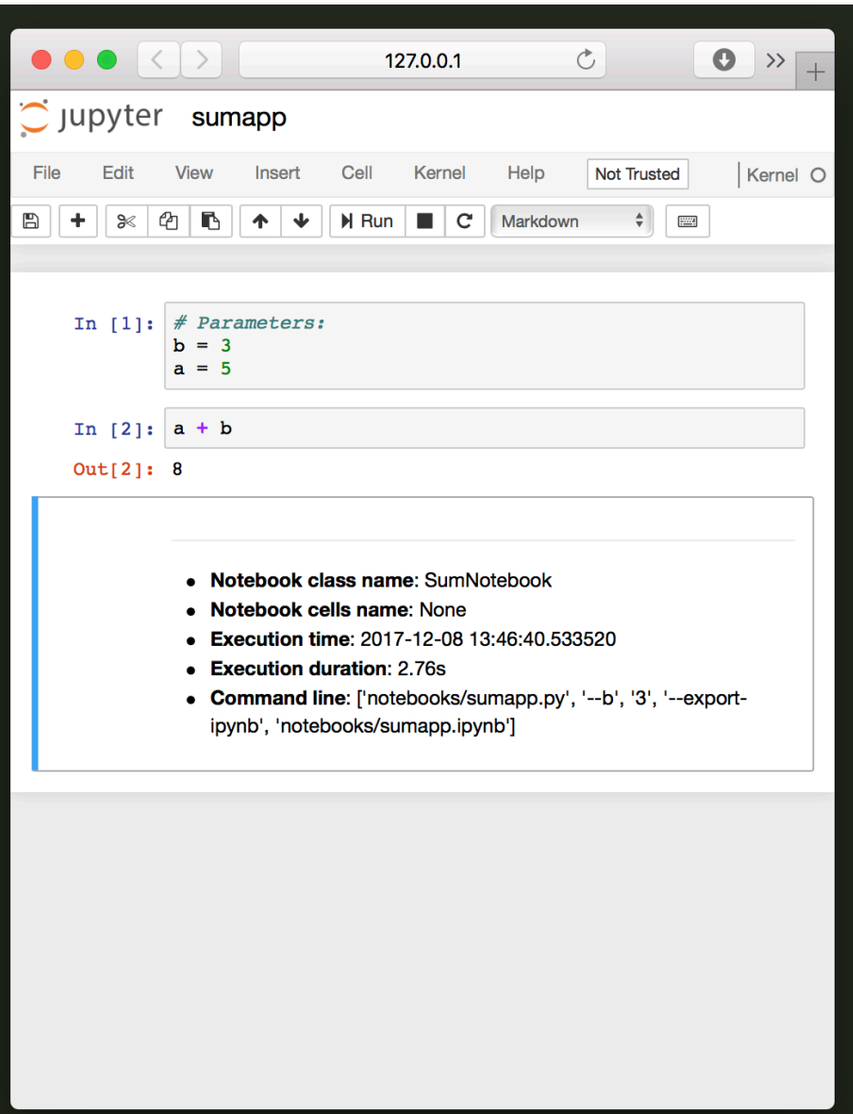
```python
def cells(a, b):
    '''
    # Sum
    '''

    a, b = int(a), int(b)

    '''
    '''

    a + b
```



Programmatic interface

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

```python
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('-')
```



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
```