Introduction to Python

Giulio Salierno

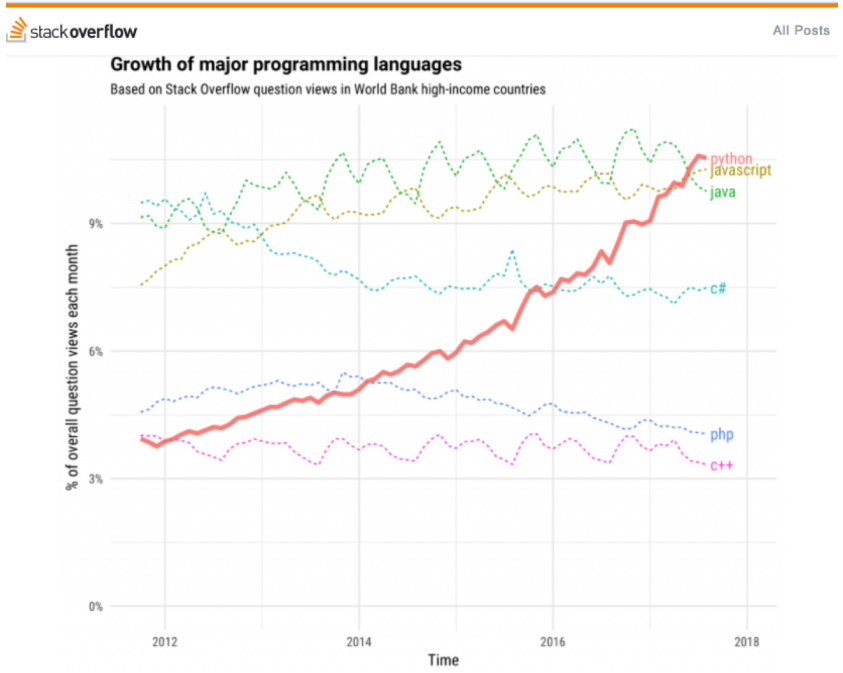
Why use Python?

- But before what is Python?
  - *Python is a programming language that blends procedural, functional, and object-oriented paradigms—*
- It blends different types of programming languages:
  - Procedural programming, based of procedures or functions (C, C++)
  - Functional programming (Lisp, Heskell)
  - Object oriented paradigms (Java)

- Python is also used as a *scripting language (bash)*
- A single language that encompass all the concepts mentioned.

Who use Python today?

- ~ 1 million of users, consist of a very active development community
- It is also included in all commercial operating systems: Linux, Windows and Mac OS (currently as Python 2.7)
- Top 5 or top 10 of most used programming languages nowadays

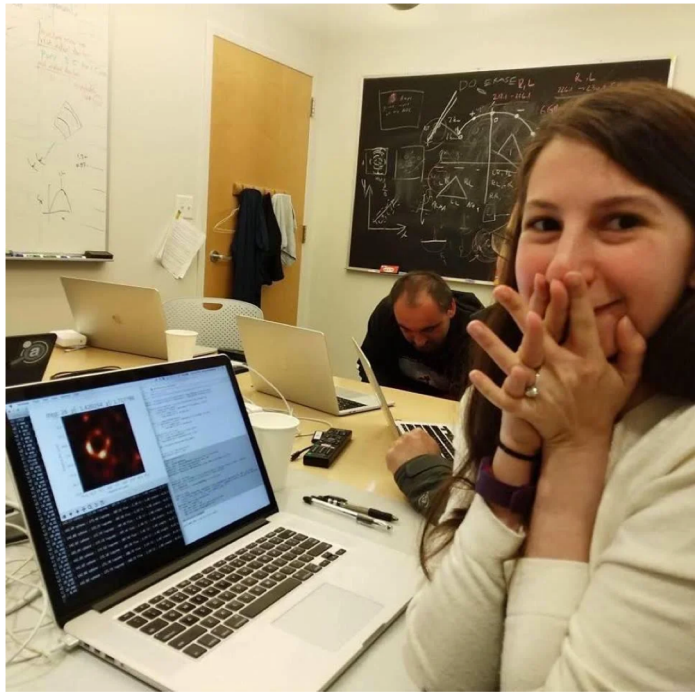


A (not) exhaustive list of companies adopting Python

- *Google* makes extensive use of Python in its web search systems.
- The popular *YouTube* video sharing service is largely written in Python.
- The *Dropbox* storage service codes both its server and desktop client software primarily in Python.
- The *Raspberry Pi* single-board computer promotes Python as its educational language.
- The NSA uses Python for cryptography and intelligence analysis.
- Intel, Cisco, Hewlett-Packard, Seagate, Qualcomm, and IBM use Python for hardware testing.
- And many many more…

What can I do with Python?

Almost everything… Python is adopted at different levels for different tasks (from web development to spacecraft control)



- *Katie Bouman* renders the first image (8.5 Petabyte data) of a black hole with Python
  - Libraries: numpy, Scipy, pandas, matplotlib

- **System Programming:** Python has a standard library to interact with Operating Systems (I/O, files, sockets, pipes, processes, threads, reg exp, data manipulation: XML, JSON etc..)
- **GUI (Graphical User Interface):** Python comes with a standard object-oriented interface to the Tk GUI API called tkinter (Tkinter in 2.X) that allows Python programs to implement portable GUIs with a native look and feel

What can I do with Python? (I)

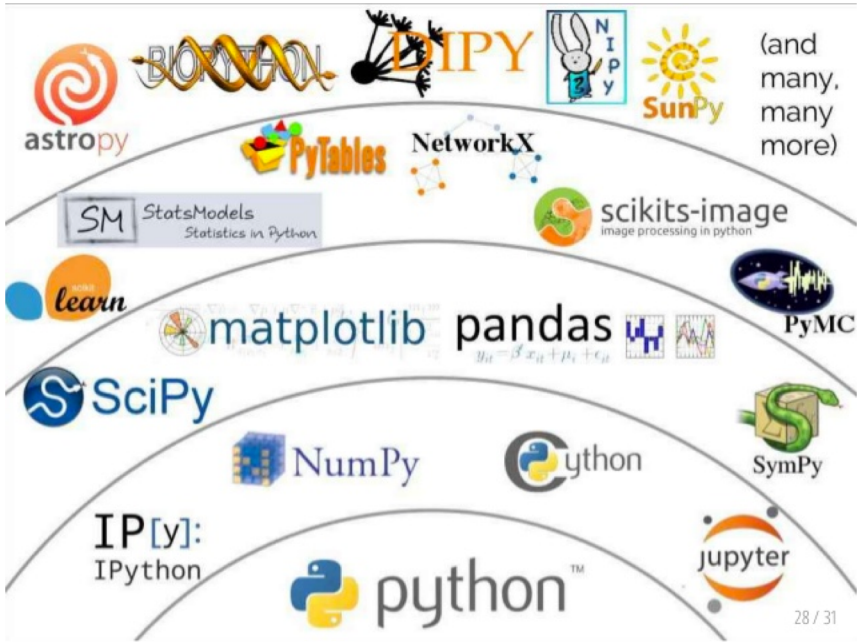
Network Programming: Python comes with standard libraries to implement several client/server architectures:

- HTTP client/server (handle GET POST request, parse URL, fetch resource)
- Parsing XML , JSON and csv data

- **Database Programming**: standard libraries to interface with common DBMS (Oracle, PostgreSQL, SQLite, MySql)
- **Rapid Prototyping:** Implement a scratch idea before a real implementation using other languages (C, C+ etc.)

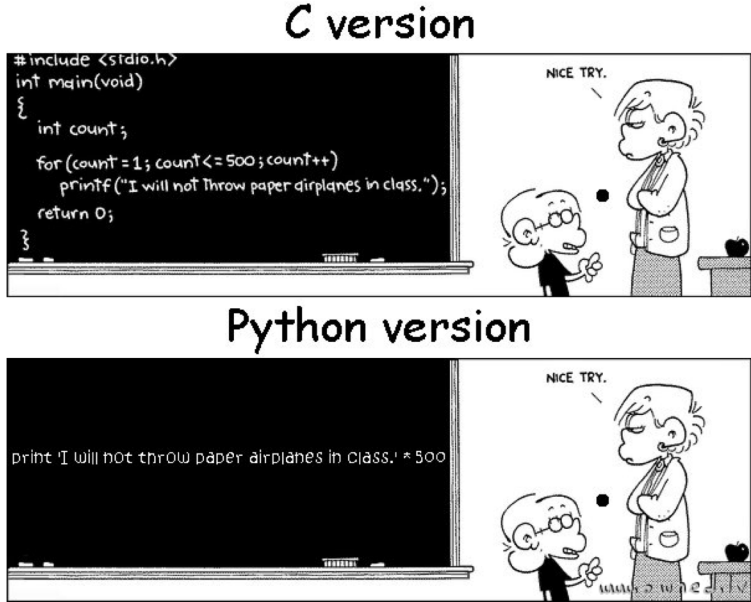
And many many more…

- **Numeric and scientific Programming**: Numpy library definies a set of matematical objects like array, matrix to simply numeric programming compared to FORTRAN and C++
- **Data Science:** Working with data is made simple thanks to:
  - NLTK: Natural language toolkit
  - Scipy: Define ML models for data analysis
  - Numpy: mathematical objects
  - Pandas: for data analysis and data structures
  - Matplotlib: visualize and plot data



Python: Strength and Weakness

- Pros:
  - It's strong OOP oriented:
    - It implements major concepts of object-oriented paradigms: polymorphism, multiple inheritance, overloading
  - It's powerful: lot of complicated tasks can be solved in a few lines of code
  - It's free: open-source and strong supported by large communities of developer
  - It's portable: python code can be executed on (almost) every operating system (from smartphone to supercomputers)
  - It's (relatively) simple to use and learn
    - No intermediate steps as other languages
    - Has a very clean and simple syntax



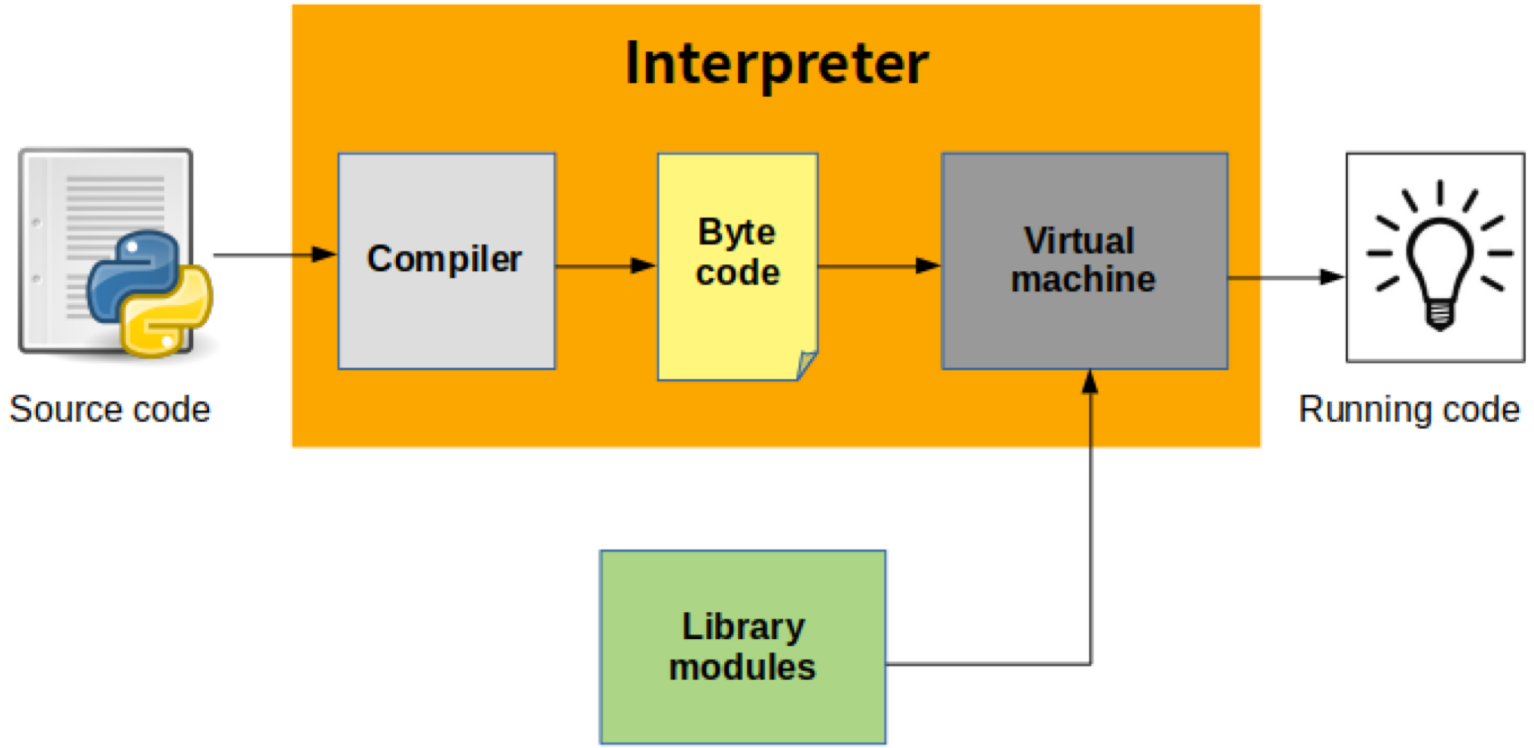
Python Weakness:
(or when i should not use Python)

- When high performances are a strong requirement
  - Python is an interpreted language (like JAVA, C#)
  - Even if CPUs are faster and faster, there are a huge
  - number of task which requires strong optimization.
- When you have to program at a low level
  - i.e., Python is not designed to design a firmware
    - (Exists Python libraries to interact with low-level components but they are very limited)

- Scalability: when you have to deal with huge amount of data Python could not be the solution (C, C++ are still preferred)

Quickstart

- What you need to start is:
  1. Python interpreter –https://www.python.org/downloads/ (Latest is 3.7.3)



Quickstart (I)

- Python programs can be executed in (mainly) two different ways:
  - **Interactively** and from **command line**

- **Interactive Prompt**
  - It is very useful to start to familiarize with python
  - To start an interactive session just type **python** into a terminal

```
MacBook-Pro-di-Giulio:~ giuliosalierno$ python
Python 2.7.10 (default, Feb 22 2019, 21:17:52)
[GCC 4.2.1 Compatible Apple LLVM 10.0.1 (clang-1001.0.37.14)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

  - It prompts interpreter information and start a new interactive session where:
    - User input a command and its output is echoed on the terminal
    - It works?

(Eventually) failed quickstart

- If no python command is found you need usually (i.e., Windows) setting up environment variables to link python command.
- Unlike most Unix systems and services, Windows does not include a system supported installation of Python

```
C:\Users\mgalarnyk>python
'python' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\mgalarnyk>jupyter notebook
'jupyter' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\mgalarnyk>conda list
'conda' is not recognized as an internal or external command,
operable program or batch file.
```

- Setting up system variables references:
  - https://medium.com/@GalarnykMichael/install-python-on-windows-anaconda-c63c7c3d1444 (anaconda)
  - https://stackoverflow.com/questions/3701646/how-to-add-to-the-pythonpath-in-windows

Getting started (Interactively)

- Running python program in interactive mode:

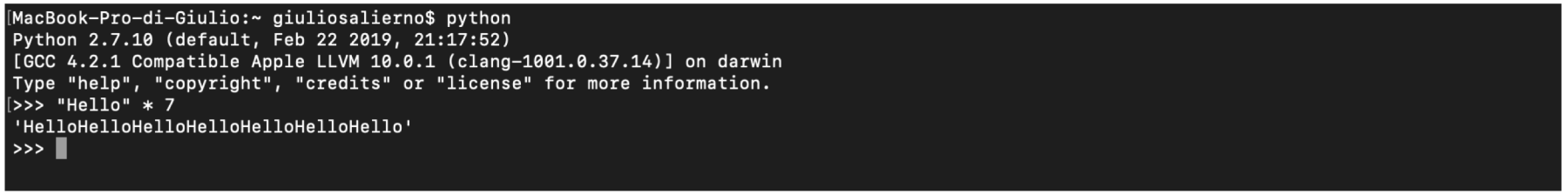
```
MacBook-Pro-di-Giulio:~ giuliosalierno$ python
Python 2.7.10 (default, Feb 22 2019, 21:17:52)
[GCC 4.2.1 Compatible Apple LLVM 10.0.1 (clang-1001.0.37.14)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print('hello world')
hello world
>>>
```

- Don’t worry about syntax (we will see it later), when coding interactively like this, you can type as many Python commands as you like, each is run immediately after it’s entered.

```
>>> print('hello world')
hello world
>>> 2**8
256
>>> variable="hello world"
>>> variable
'hello world'
>>>
```

Interactive mode

- Why is it useful? Because you can experiment it does not save the code in a file you can use it if you have a doubt or you don’t remember the behaviour of a python command.
- It is a great place to experiment and test it is not intended to develop a complete python program



- The feedback you receive from the prompt it’s usually the most immediate suggestion on what a command does.
- If not satysfing:
  - https://docs.python.org/3/
  - Everything related to the language (libraries, syntax, data types, and so on)

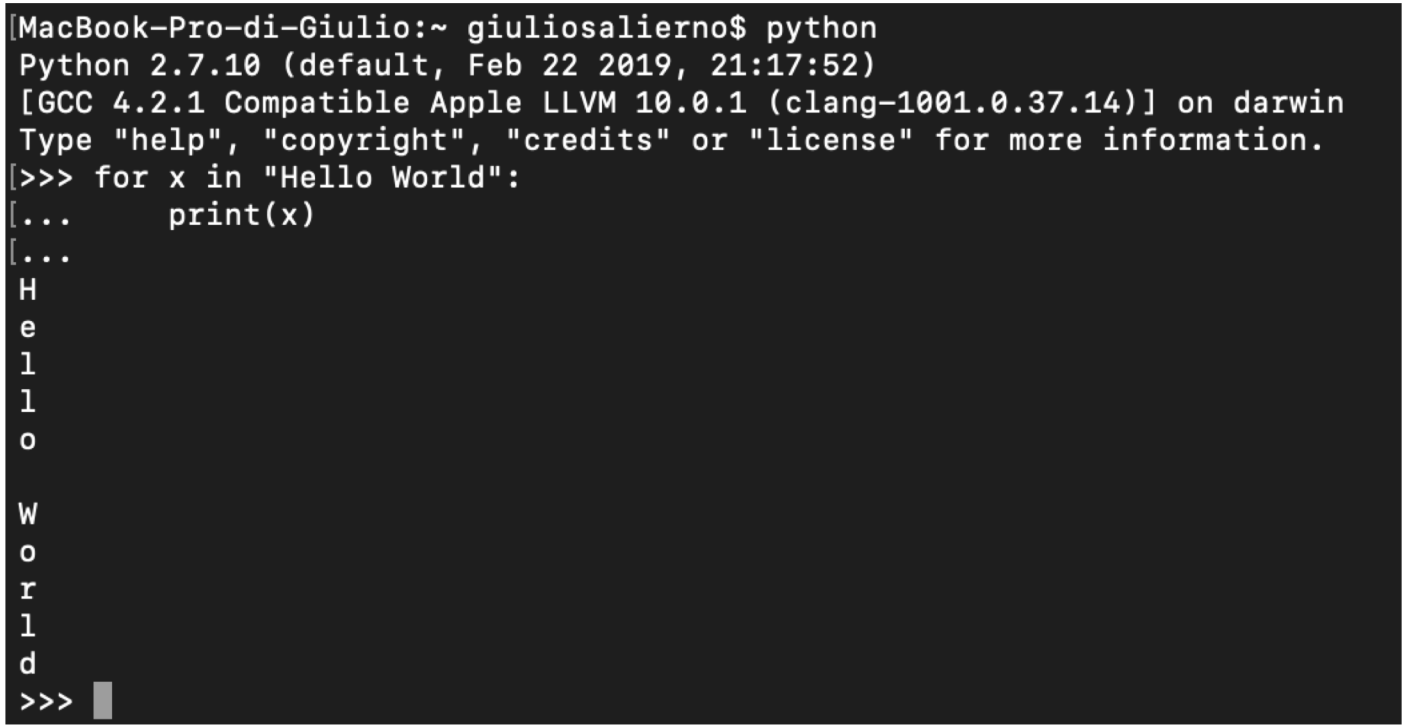
Interactive mode (I)

- Testing python commands:
- Example: *import* is a reserverd keyword in python for indicating the interpreter to load a library (called **module**)
- *os* is the name of a standard python library to interact with the underlying OS from a python interpreter.

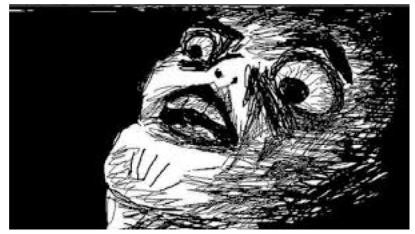
```
MacBook-Pro-di-Giulio:~ giuliosalierno$ python
Python 2.7.10 (default, Feb 22 2019, 21:17:52)
[GCC 4.2.1 Compatible Apple LLVM 10.0.1 (clang-1001.0.37.14)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> os.getcwd()
'/Users/giuliosalierno'
>>>
```

Interactive mode – compound statement

- While ignored in Python scripts, terminate multiline compound statements like loops and tests *for if* at the interactive prompt with a blank line. In other words, you must press the Enter key twice, to terminate the whole multiline statement and then make it run



Interactive Mode - (II)

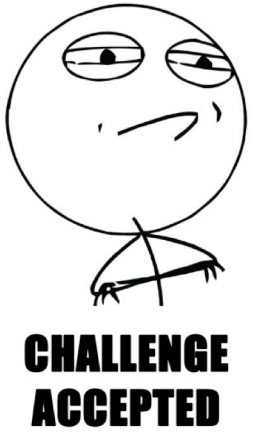


- Interactive mode is great for live testing and experimenting the code, but it does not store your code anywhere.
- If you want to re-test a snippet of code you must rewrite it!
  - This is great for testing but not suitable for a complete python program

- Solution: Store lines of python code in a file
  so it can be executed everytime is needed.

- Python files are often called modules or scripts (as os seen before)
- The terminology can vary but they refer to the same thing
- A python module containes python code

Command line mode:



- Every python module as an extension: *.py*
- Open your favourite editor (vi, Notepad, IDLE editor)

  and write your first python program as follows:

```
import sys #import module

print(sys.platform)

print (2**12)

variable="hello world"

print(variable)


print(variable * 4)
```

Save it as: *script.py* in your working directory (if not exist create a new one)

- Run the script located in your working directory: *python* script.py
- Check the output

Unix-like syntax: shabang #!/usr/bin/python

- Altought not required on Windows, as a best-practice it is required to add the shabang as the first line of a python script.
- The line #!/usr/bin/python expressed as the first line tells the terminal where the python interpreter lives.

- ‘#’ is used to express comments but since it resides on the first line of a script it is special interpreted
- The UNIX-shell will execute the script using the python interpreter specified in the first line

- On some Unix systems, you can avoid hardcoding the path to the Python interpreter in your script file by writing the special first-line comment like this:

```
#!/usr/bin/env python
print(”hello” + “this is a” + ”unix-compliant” + “python” + “script”) # + means concat
```

- This command is ignored if the script is executed on Windows, but guaarantes portability of python programs.

What IDE use from now?

- We have seen two of most common methods to run Python
- Other methods exist like using IDLE which is a standard IDE that comes with python.
- PyCarm a professional IDE for Developers that include other tools

like versioning and a powerful debugger (indicated if you have to manage very large project).

- What I recommend: simple text-editor: vi (UNIX) or Sublime Text (Notepad++)

- Sublime Text is a lightweight editor with a GUI it can be used as a standard IDE for almost every modern language
- Command line to execute your python programs

What use from now? (II) - Juypter

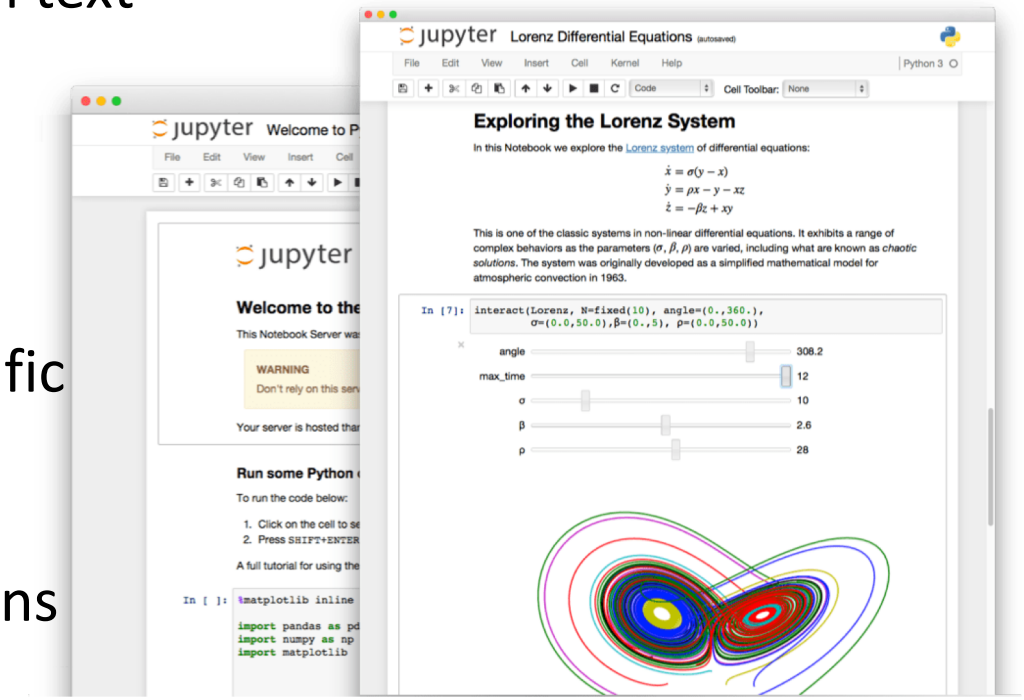
- Juypter is web-app to manage notebooks via browser.

- A notebook contains python code mixed with text

- Snippet of code contained in a notebook
- can be executed indipendently

- Juypter notebooks are heavily used in scientific
- programming as well as in data science

- They are self-explanatory: a notebook contains
- Code, text,image, charts, graphs



When to use Juypter?

- It comes with *Anaconda* just type: *conda install juypter*
  - Run it with by typing: *juypter* from the command line

- *It is delivered as a cloud service by Google:*
  - https://colab.research.google.com/notebooks/welcome.ipynb
  - *Juypter Notebook will be stored on Google Drive*

- I will use juypter notebook 70% of times – why not 100%?

- Because it particular designed for manage one-file python projects.
- For projects containing multile files i will launch programs from command line

Juypter – Notebook example